US009391857B2

(12) **United States Patent**
Umanesan

(10) **Patent No.: US 9,391,857 B2**
(45) **Date of Patent: *Jul. 12, 2016**

(54) **SCHEDULING REQUESTS FOR DATA TRANSFERS IN A MULTI-DEVICE STORAGE SYSTEM**

(71) Applicant: **Seagate Technology LLC**, Cupertino, CA (US)

(72) Inventor: **Ganesan Umanesan**, Guildford (GB)

(73) Assignee: **Seagate Technology LLC**, Cupertino, CA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

This patent is subject to a terminal disclaimer.

(21) Appl. No.: **14/550,516**

(22) Filed: **Nov. 21, 2014**

(65) **Prior Publication Data**

US 2015/0149622 A1 May 28, 2015

Related U.S. Application Data

(63) Continuation of application No. 13/193,005, filed on Jul. 28, 2011, now Pat. No. 8,909,764.

(51) **Int. Cl.**

H04L 12/26	(2006.01)
H04L 29/08	(2006.01)
G06F 15/173	(2006.01)

(52) **U.S. Cl.**

CPC ***H04L 43/08*** (2013.01); ***H04L 67/16*** (2013.01); ***H04L 67/325*** (2013.01); ***H04L 67/327*** (2013.01)

(58) **Field of Classification Search**

CPC H04L 43/08; H04L 67/325; H04L 67/327; H04L 67/1004; H04L 2012/5681; H04L 2012/5678; H04L 43/16; H04L 29/08153; G06F 9/489; G06F 9/4881

USPC 709/224–226, 232, 233, 238, 240, 241, 709/200; 718/102–103; 710/18, 39; 370/395.4, 42, 412

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,721,726	A	2/1998	Kurnick et al.
6,078,998	A	6/2000	Kamel et al.

(Continued)

FOREIGN PATENT DOCUMENTS

EP	1083709	3/2001
WO	03/094451	11/2003

OTHER PUBLICATIONS

Leonidas Georgiadis, "Optimal Multiplexing on a Single Link: Delay and Buffer Requirements," IEEE Transactions on Information Theory, vol. 47. No. 5, pp. 1518-1535 (Sep. 1997).

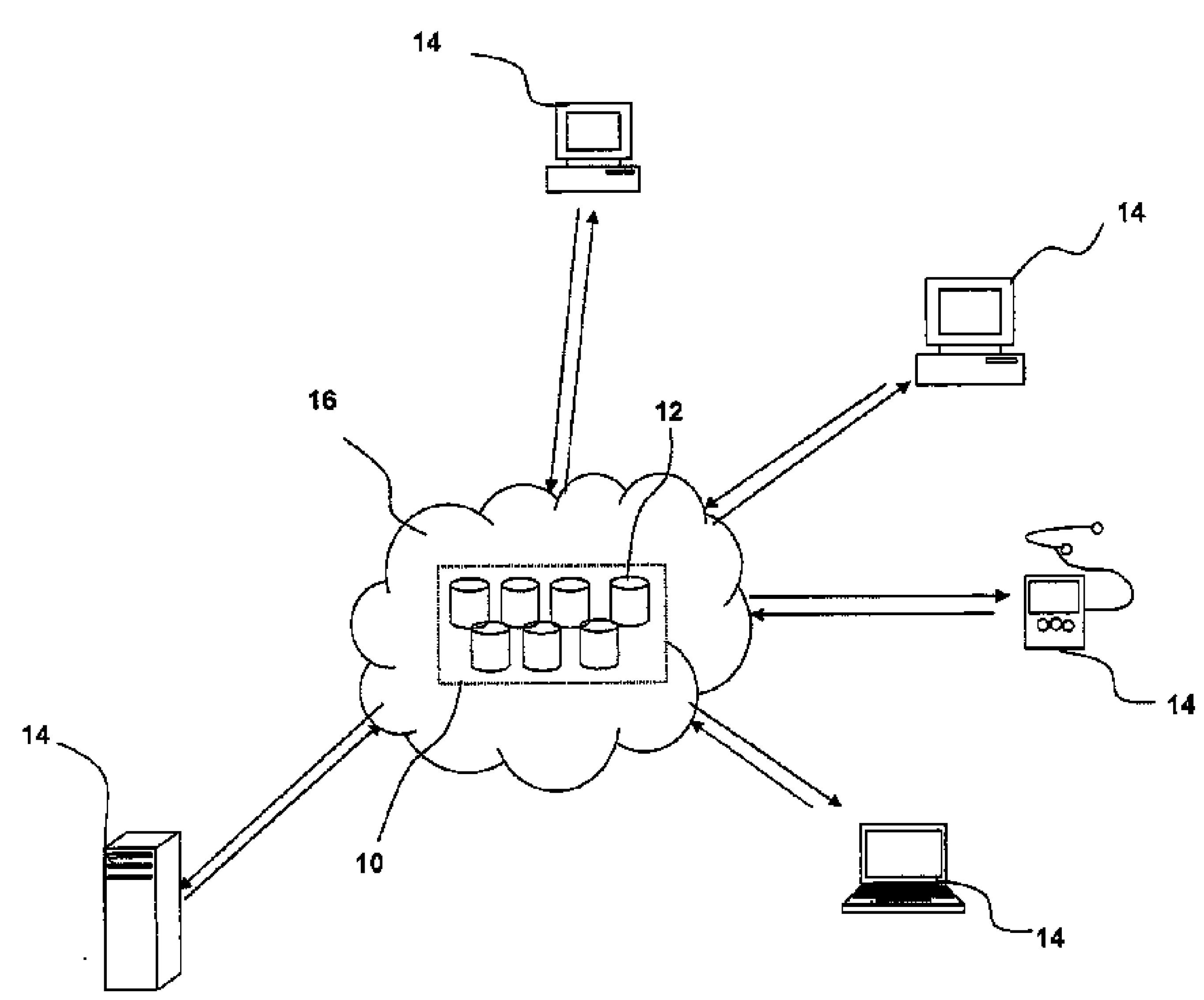
(Continued)

Primary Examiner — Alina N Boutah
(74) *Attorney, Agent, or Firm* — Hall Estill Attorneys at Law

(57) **ABSTRACT**

Apparatus and method for scheduling requests for data transfers in a multi-device storage system. In some embodiments, a system includes at least one server coupled to a pool of storage devices to transfer data from the storage devices to client devices responsive to requests. A request scheduler is adapted to receive into a memory a plurality of requests each having a service identifier (ID) and a payload size, to set a deadline for each request responsive to the service ID and the payload size, to forward the requests to the server for processing in an order based on service ID and, responsive to the deadline being reached for a selected request, to advance the selected request for immediate processing by the server.

20 Claims, 17 Drawing Sheets



(56) **References Cited**

U.S. PATENT DOCUMENTS

6,182,120	B1	1/2001	Beaulieu et al.	
6,594,263	B1	7/2003	Martinsson	
6,728,270	B1 *	4/2004	Meggers et al.	370/514
6,965,965	B2	11/2005	Espeseth et al.	
7,061,918	B2	6/2006	Duckering et al.	
7,145,871	B2	12/2006	Levy et al.	
7,260,703	B1	8/2007	Moore et al.	
7,454,457	B1 *	11/2008	Lowery et al.	709/203
7,577,763	B1 *	8/2009	Beaman	710/6
7,606,927	B2 *	10/2009	Tasman et al.	709/232
7,917,903	B2	3/2011	Lumb et al.	
8,266,289	B2 *	9/2012	Saha et al.	709/226
9,130,969	B2 *	9/2015	Umanesan	
2002/0031086	A1 *	3/2002	Welin	370/229
2003/0074256	A1 *	4/2003	LaCroix	705/14
2003/0154272	A1 *	8/2003	Dillon et al.	709/223
2003/0179703	A1	9/2003	Levy et al.	
2003/0202517	A1 *	10/2003	Kobayakawa et al.	370/395.4
2004/0064557	A1	4/2004	Karnik et al.	
2004/0267982	A1	12/2004	Jackson et al.	
2005/0157752	A1 *	7/2005	Takase et al.	370/468
2005/0163048	A1	7/2005	Arora et al.	
2006/0233108	A1	10/2006	Krishnan	
2007/0104099	A1	5/2007	Mutnuru et al.	
2007/0286097	A1 *	12/2007	Davies	370/255
2008/0120424	A1 *	5/2008	Deshpande	709/230
2008/0189350	A1	8/2008	Vasa et al.	
2008/0273533	A1 *	11/2008	Deshpande	370/392
2009/0327533	A1 *	12/2009	Kallam et al.	710/25
2011/0019568	A1 *	1/2011	Kim et al.	370/252
2011/0099204	A1 *	4/2011	Thaler	707/797
2011/0158182	A1 *	6/2011	Biton et al.	370/329
2012/0096175	A1	4/2012	Kourkouzelia et al.	
2012/0290707	A1 *	11/2012	Ennis et al.	709/224
2013/0013577	A1	1/2013	Fee et al.	
2014/0053160	A1	2/2014	Fee et al.	
2014/0059551	A1 *	2/2014	Umanesan	718/102

OTHER PUBLICATIONS

Matthew Andrews et al., “Minimizing End-to-End Delay in High Speed Networks with a Simple Coordinated Schedule,” IEEE, pp. 380-388 (1999).

Pawan Goyal et al., “Start-Time Fair Queueing: A Scheduling Algorithm for Integrated Services Packet Switching Networks,” IEEE/ACM Transactions on Networking, vol. 5, No. 5, pp. 690-704 (Oct. 1997).

Rene L. Cruz, “A Calculus for Network Delay, Part I: Network Elements in Isolation,” IEEE Transactionson Information Theory, vol. 37, No. 1, pp. 114-131 (Jan. 1991).

Jorg Liebeherr et al., “Exact Admission Control for Networks with a Bounded Delay Service,” IEEE/ACM Transactions on Networking, vol. 4, No. 6, pp. 885-901 (Dec. 1996).

Arshad Iqbal et al., “Dynamic Queue Deadline First Scheduling Algorithm for Soft Real Time Systems,” IEEE—2005 International Conference on Emerging Technologies, pp. 346-351 (Sep. 17-18, 2005 Islamabad).

* cited by examiner

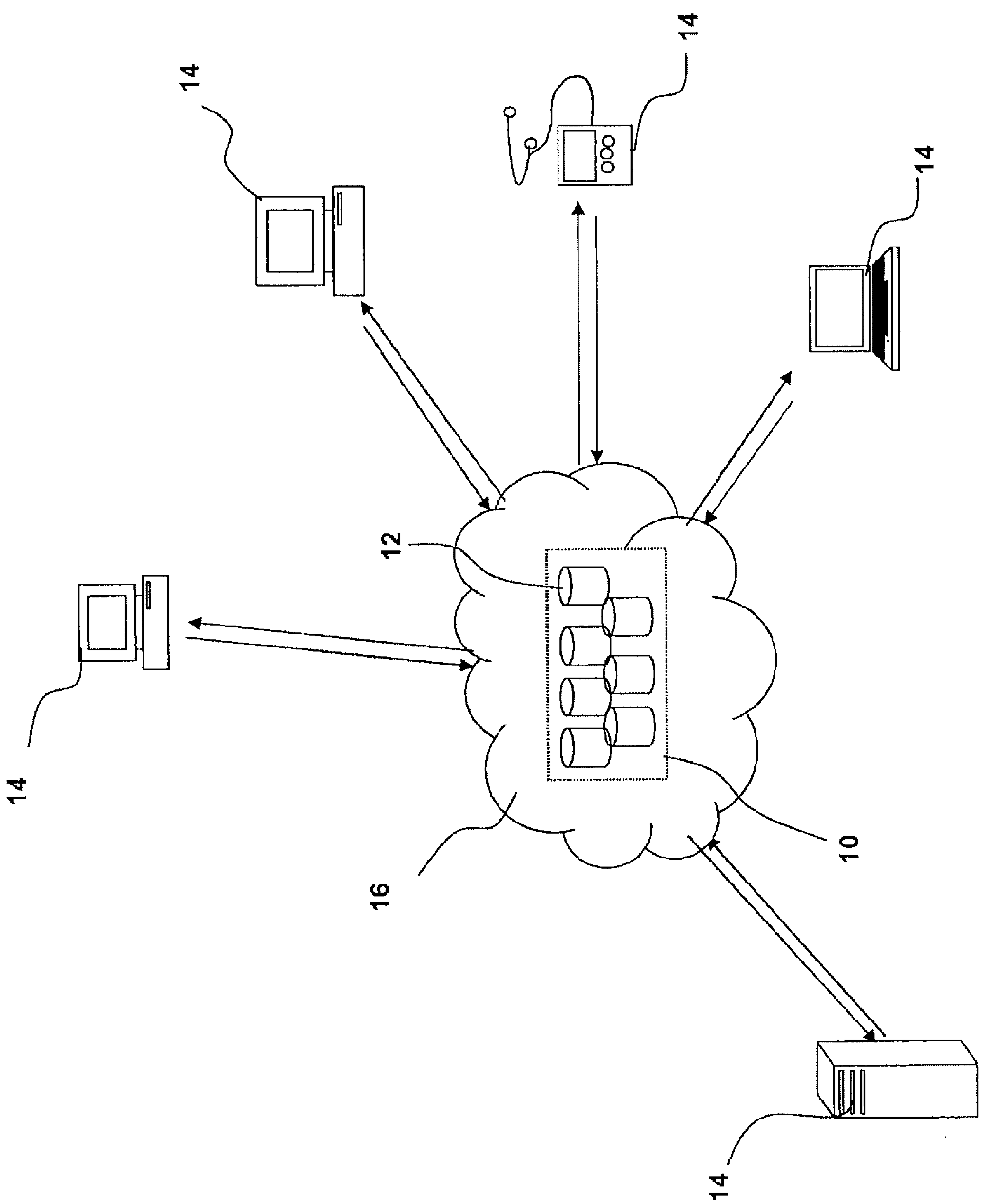


Fig. 1

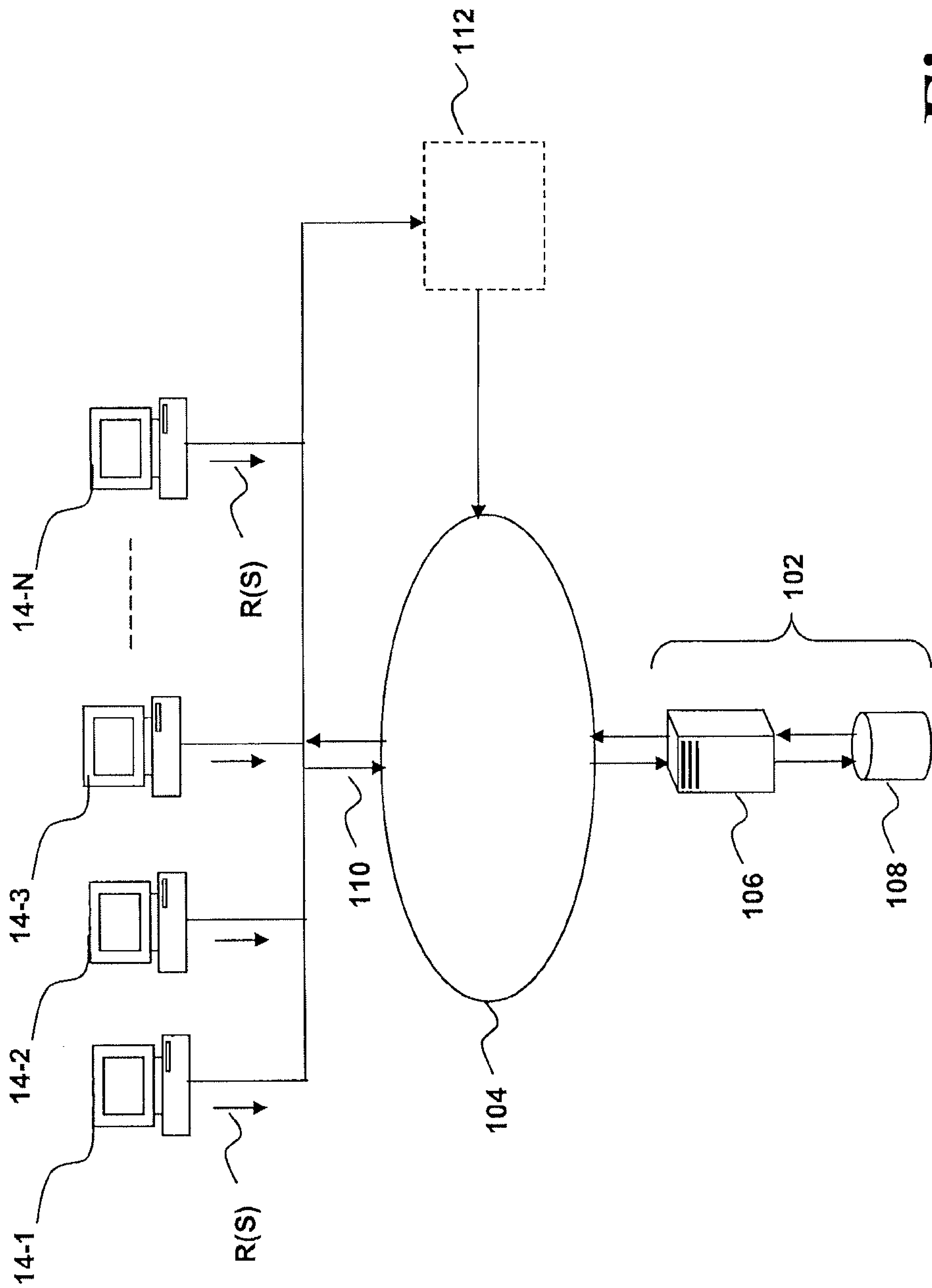


Fig. 2

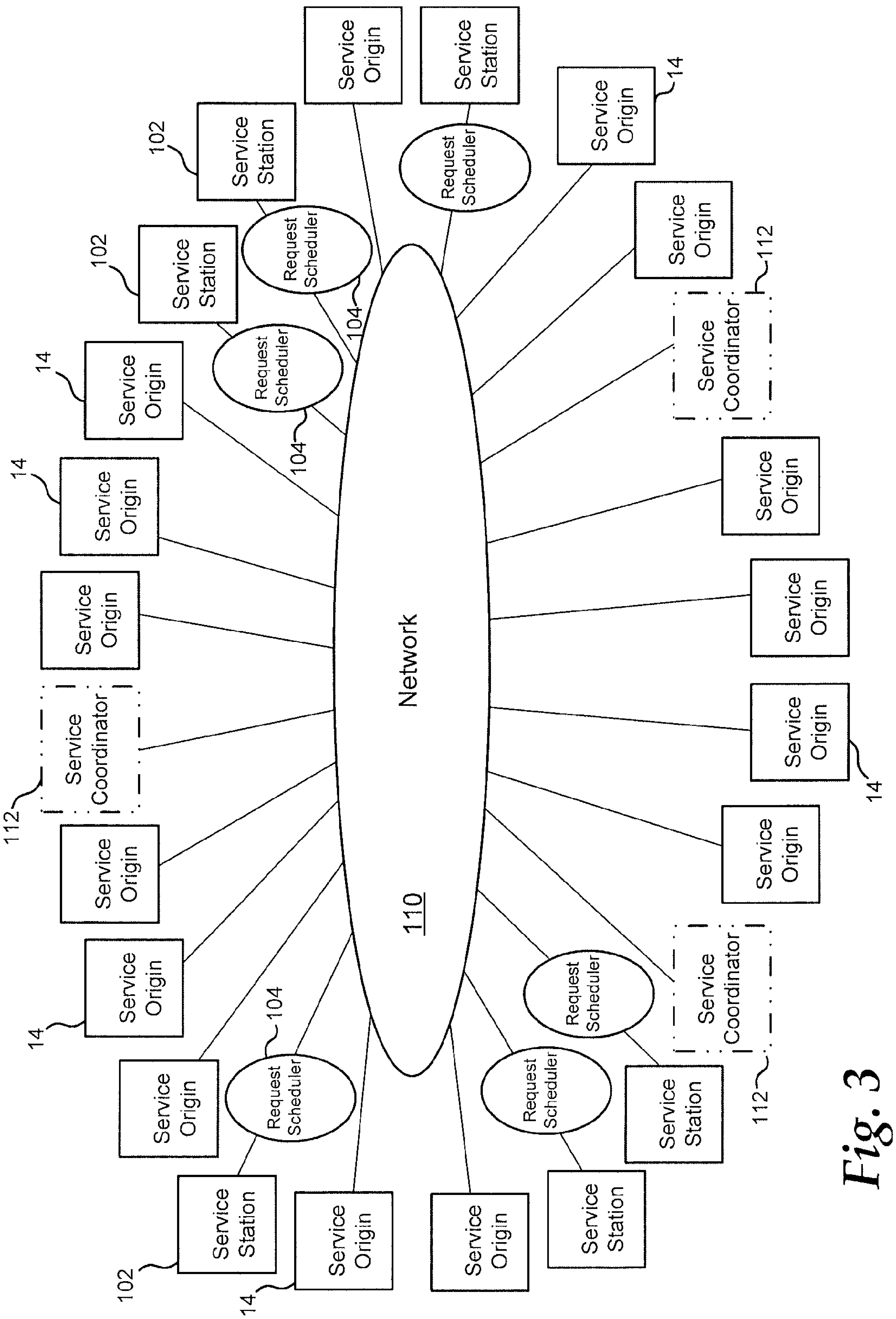

Network
110
Service Origin
Service Station
Request Scheduler
Service Coordinator
14
102
104
112

Fig. 3

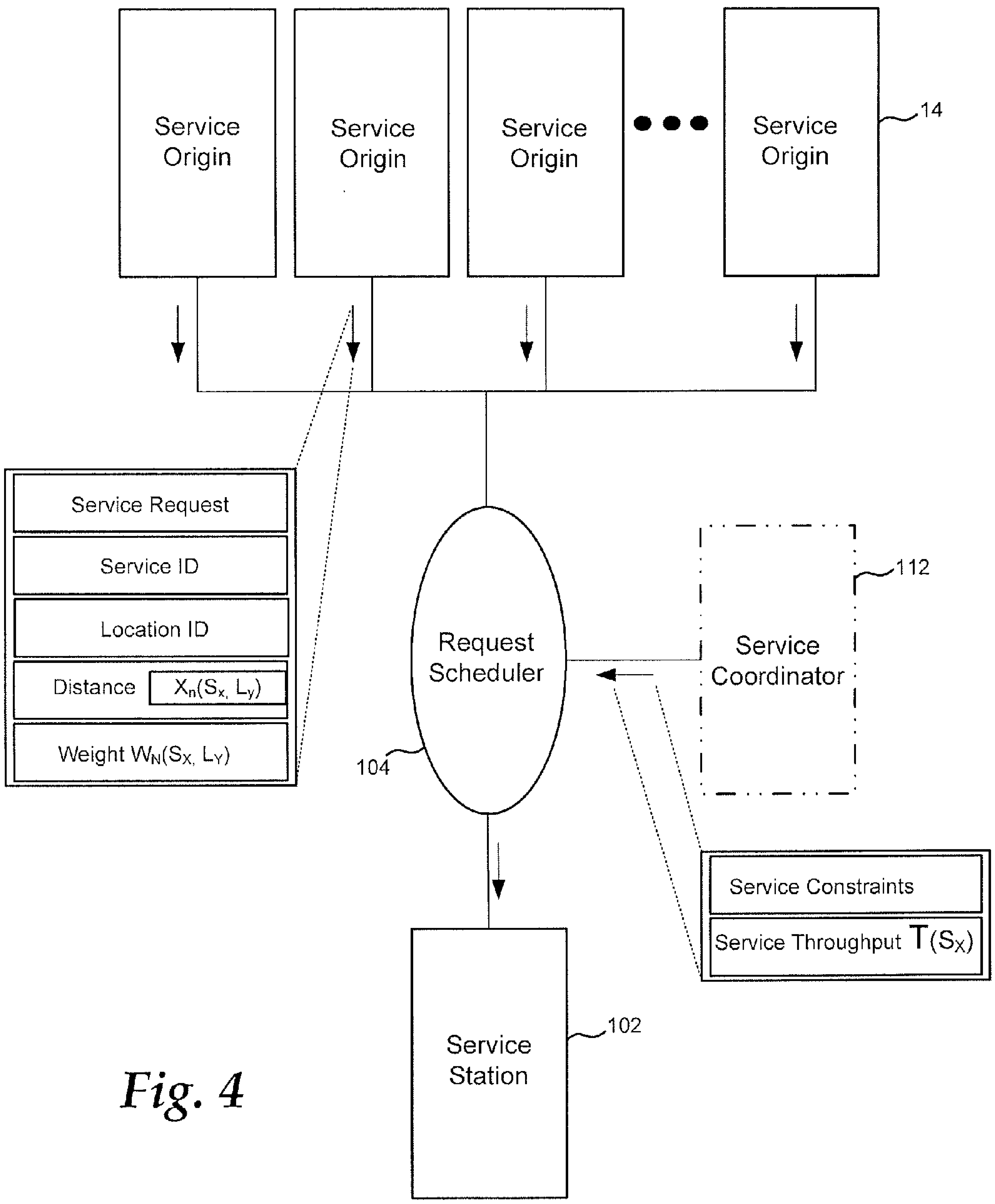

Service Origin
Service Origin
Service Origin
Service Origin
14
Service Request
Service ID
Location ID
Distance $X_n(S_x, L_y)$
Weight $W_N(S_X, L_Y)$
Request Scheduler
104
Service Coordinator
112
Service Constraints
Service Throughput $T(S_X)$
Service Station
102

Fig. 4

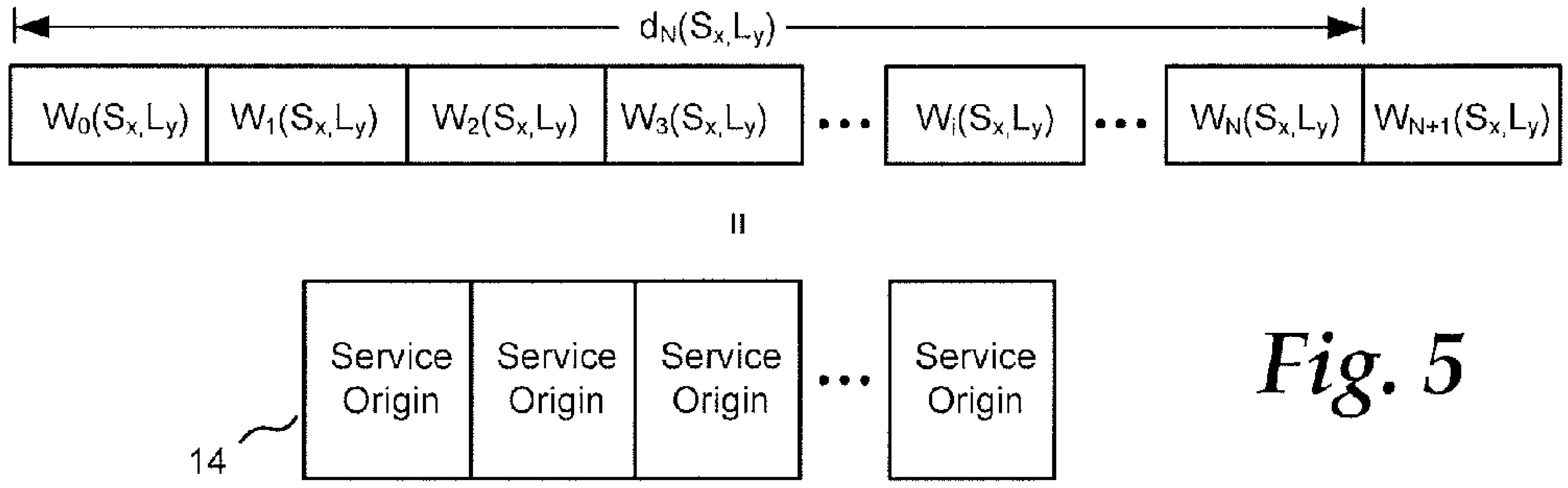

$d_N(S_x, L_y)$
$W_0(S_x, L_y)$
$W_1(S_x, L_y)$
$W_2(S_x, L_y)$
$W_3(S_x, L_y)$
$W_i(S_x, L_y)$
$W_N(S_x, L_y)$
$W_{N+1}(S_x, L_y)$
II
Service Origin
Service Origin
Service Origin
Service Origin
14

Fig. 5

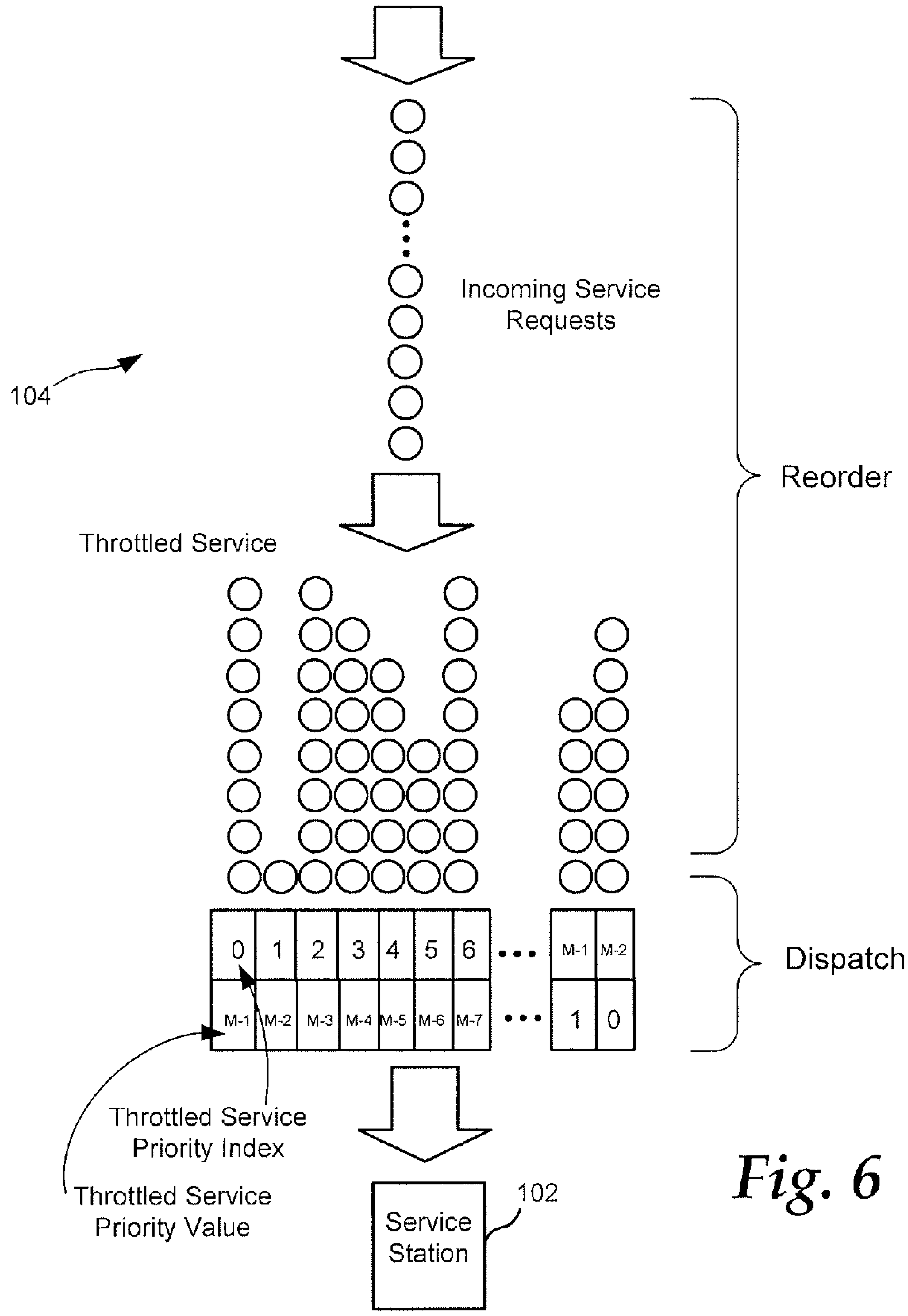

Incoming Service Requests
104
Reorder
Throttled Service
0
1
2
3
4
5
6
M-1
M-2
M-1
M-2
M-3
M-4
M-5
M-6
M-7
1
0
Dispatch
Throttled Service Priority Index
Throttled Service Priority Value
Service Station
102

Fig. 6

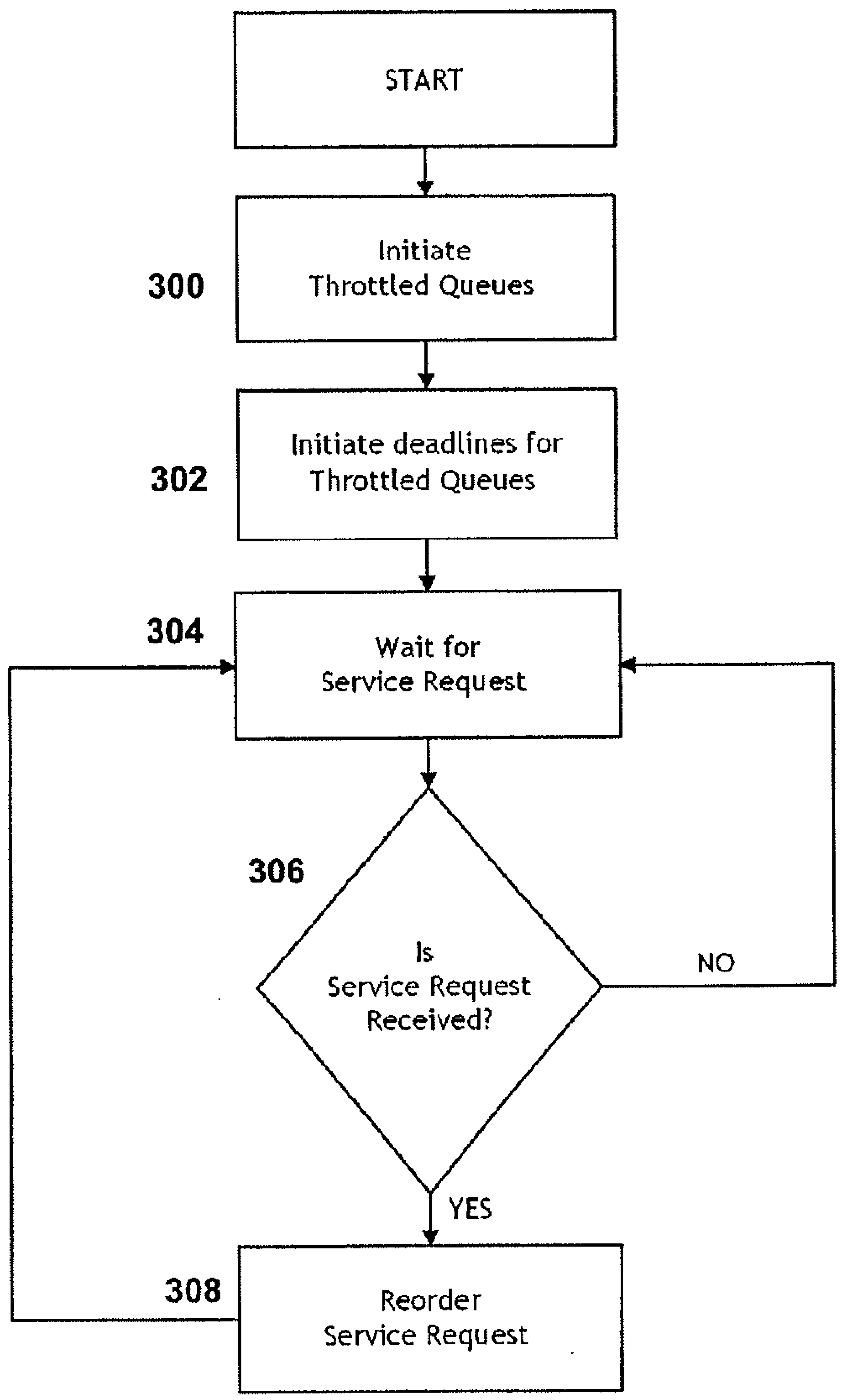

START
Initiate
Throttled Queues
300
Initiate deadlines for
Throttled Queues
302
304
Wait for
Service Request
306
Is
Service Request
Received?
NO
YES
308
Reorder
Service Request

Fig. 7

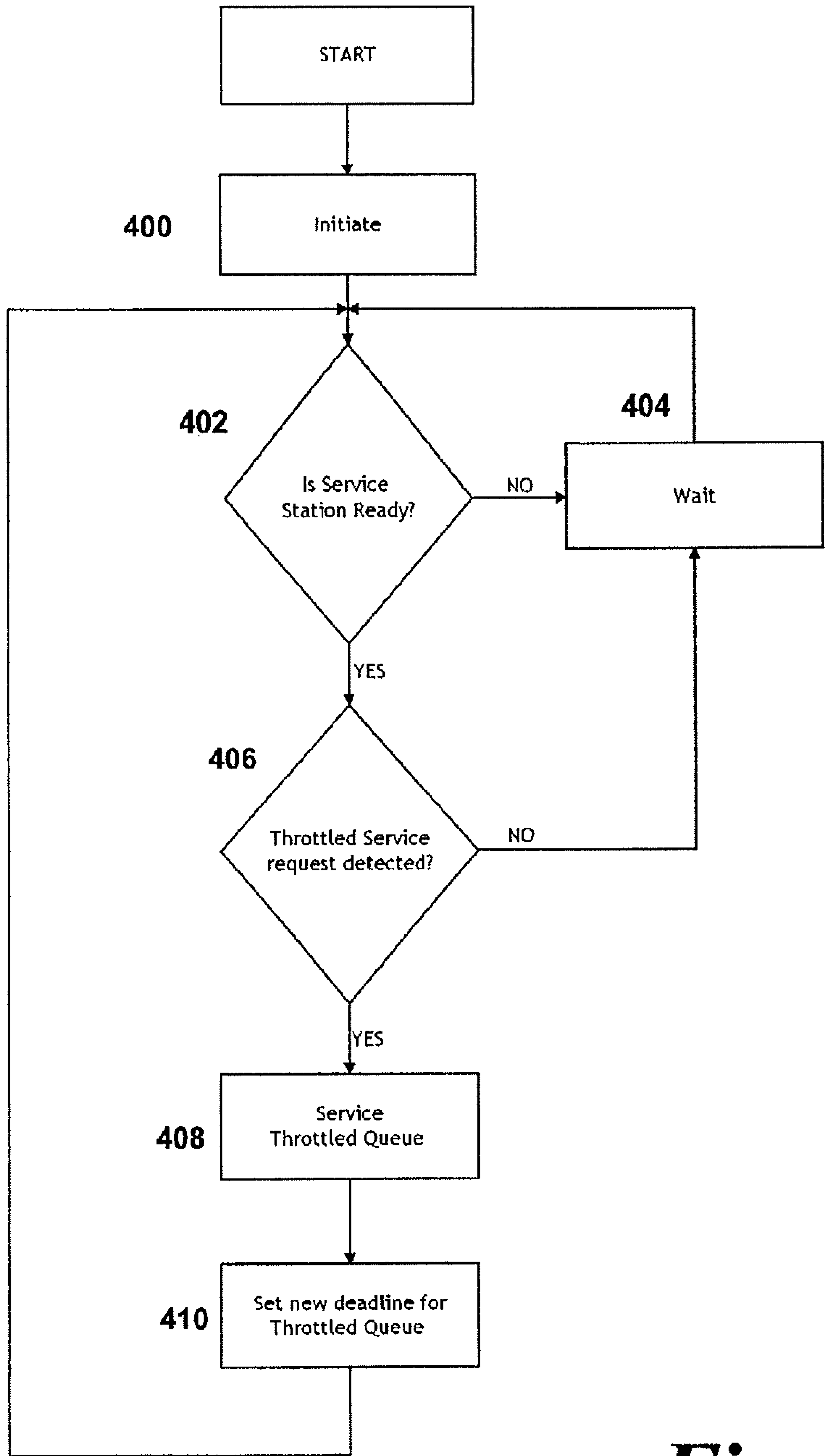

START
400
Initiate
402
Is Service Station Ready?
NO
404
Wait
YES
406
Throttled Service request detected?
NO
YES
408
Service Throttled Queue
410
Set new deadline for Throttled Queue

Fig. 8

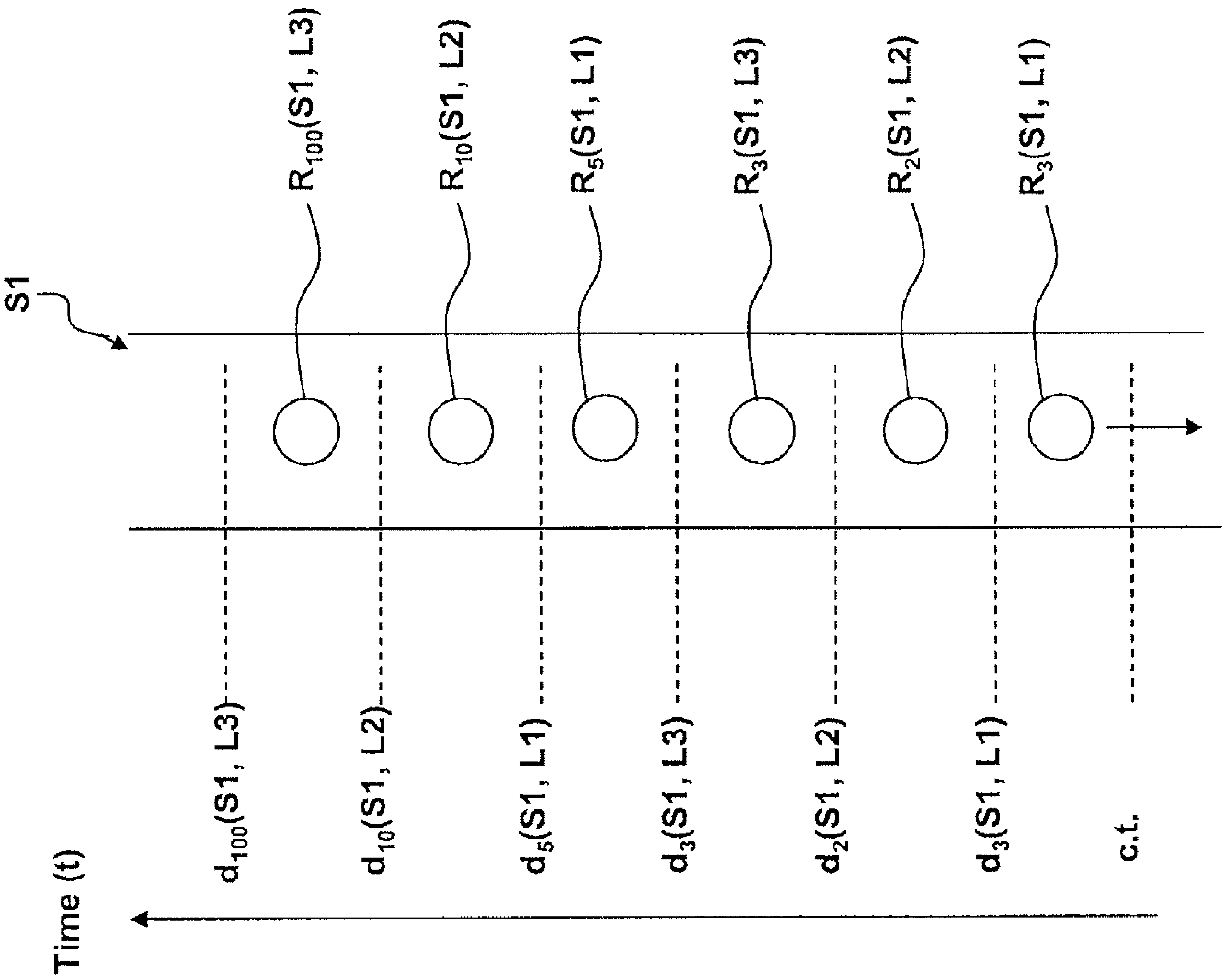

S1
R_{100}(S1, L3)
R_{10}(S1, L2)
R_5(S1, L1)
R_3(S1, L3)
R_2(S1, L2)
R_3(S1, L1)
d_{100}(S1, L3)
d_{10}(S1, L2)
d_5(S1, L1)
d_3(S1, L3)
d_2(S1, L2)
d_3(S1, L1)
c.t.
Time (t)

Fig. 10

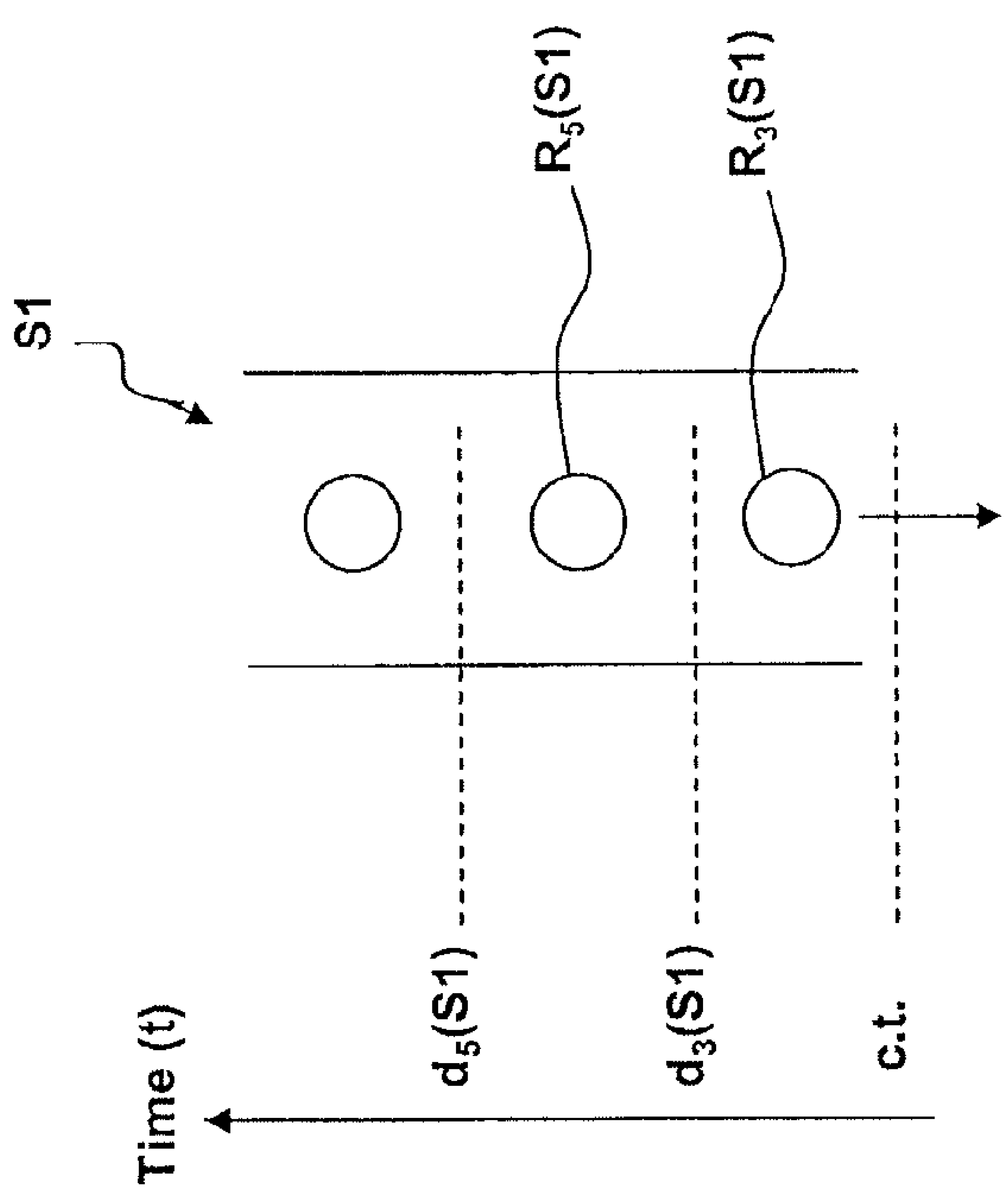

S1
R_5(S1)
R_3(S1)
d_5(S1)
d_3(S1)
c.t.
Time (t)

Fig. 9

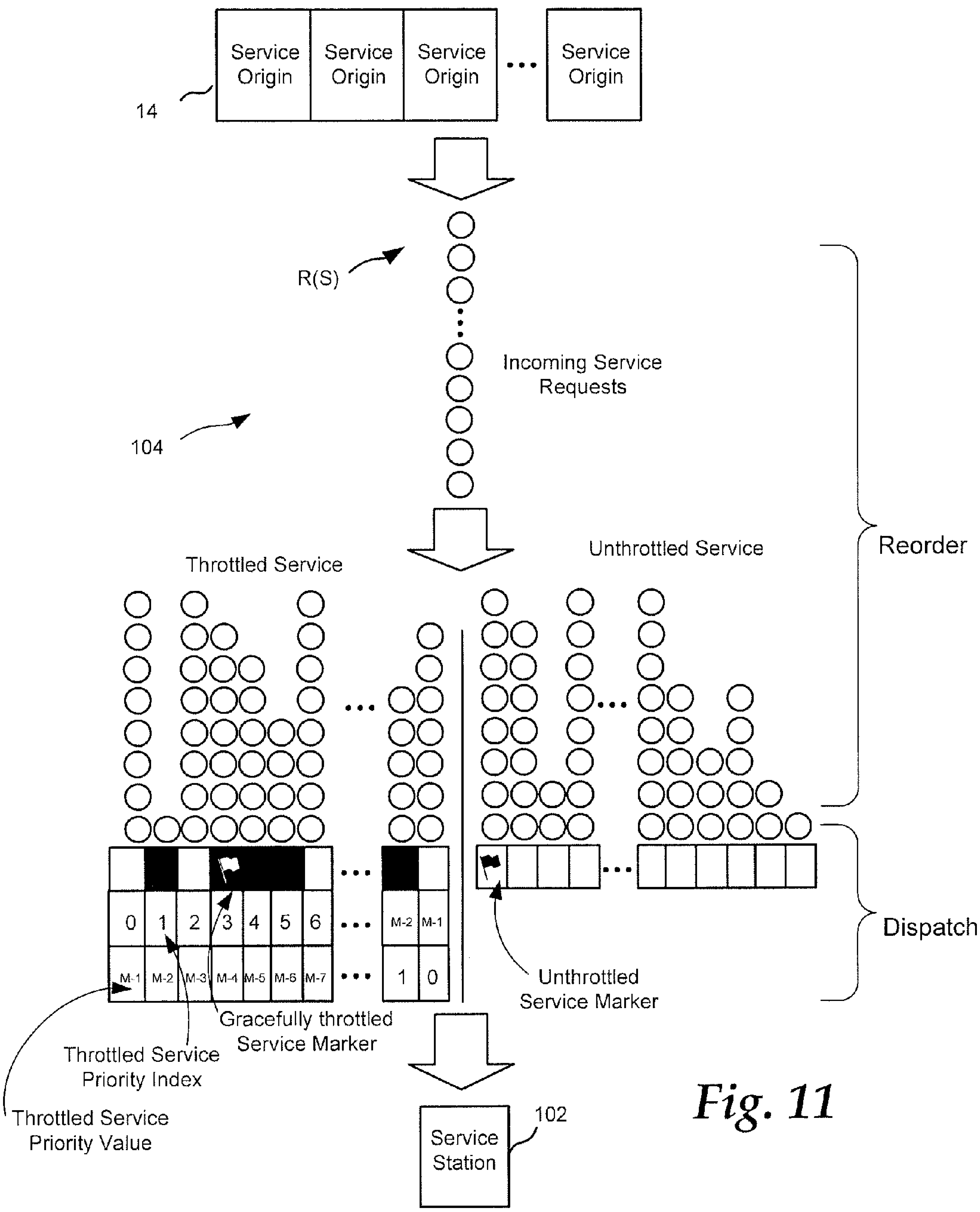

Service Origin
Service Origin
Service Origin
Service Origin
14
R(S)
Incoming Service Requests
104
Reorder
Throttled Service
Unthrottled Service
0
1
2
3
4
5
6
M-2
M-1
M-1
M-2
M-3
M-4
M-5
M-6
M-7
1
0
Dispatch
Unthrottled Service Marker
Gracefully throttled Service Marker
Throttled Service Priority Index
Throttled Service Priority Value
Service Station
102

Fig. 11

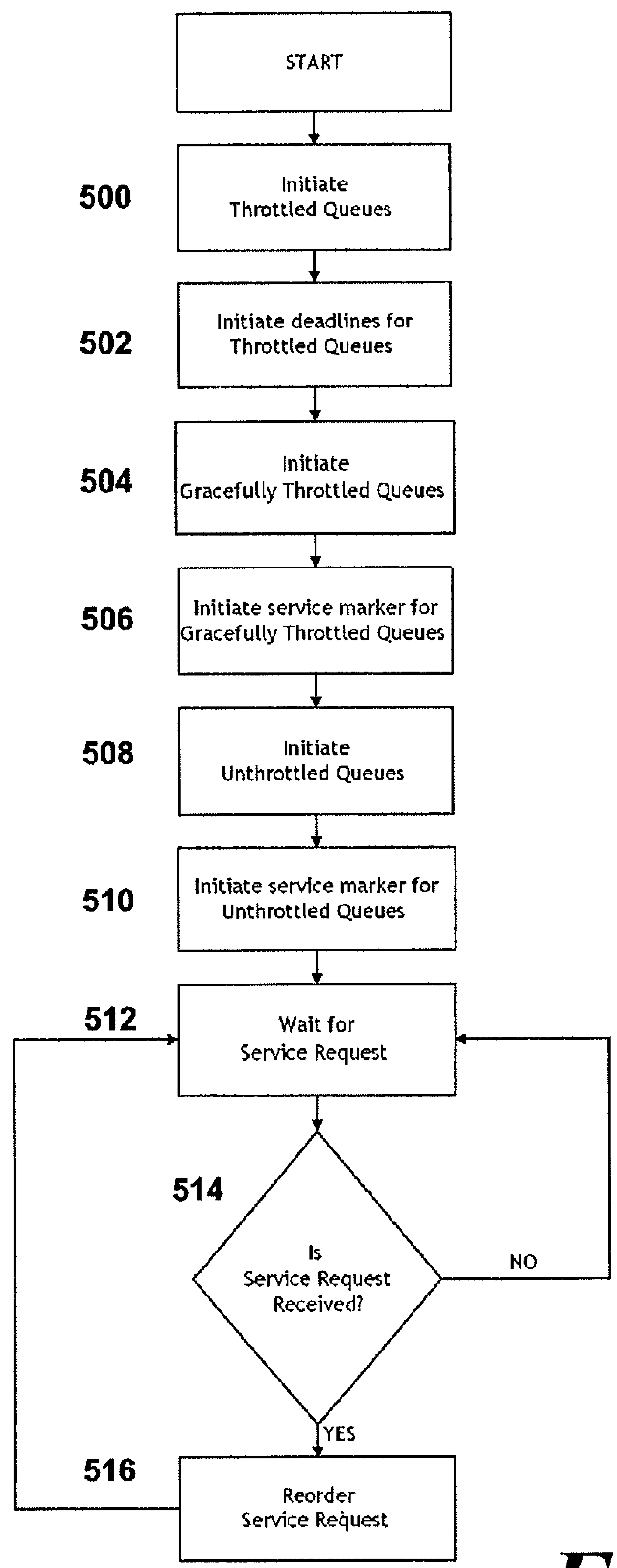
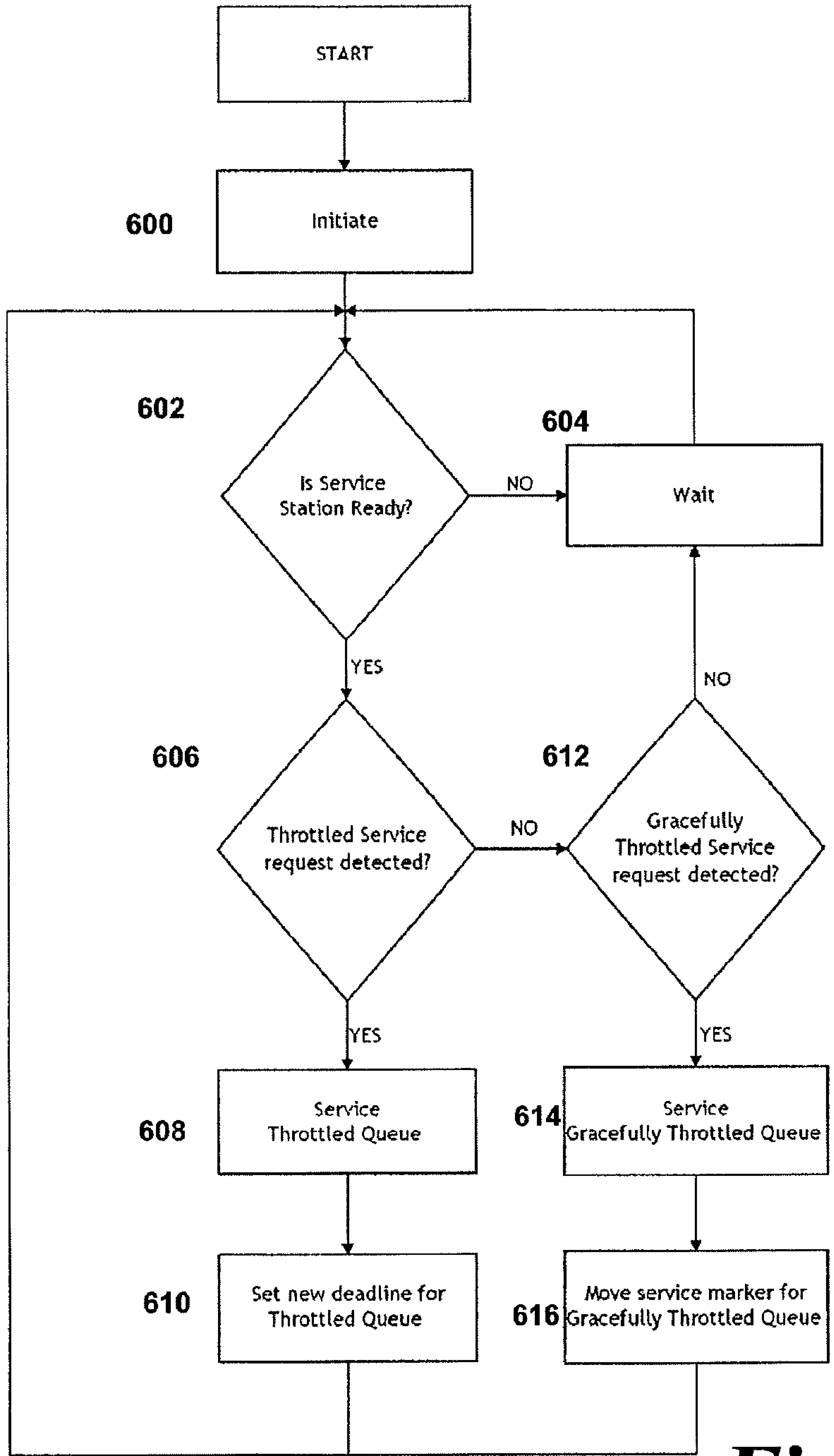
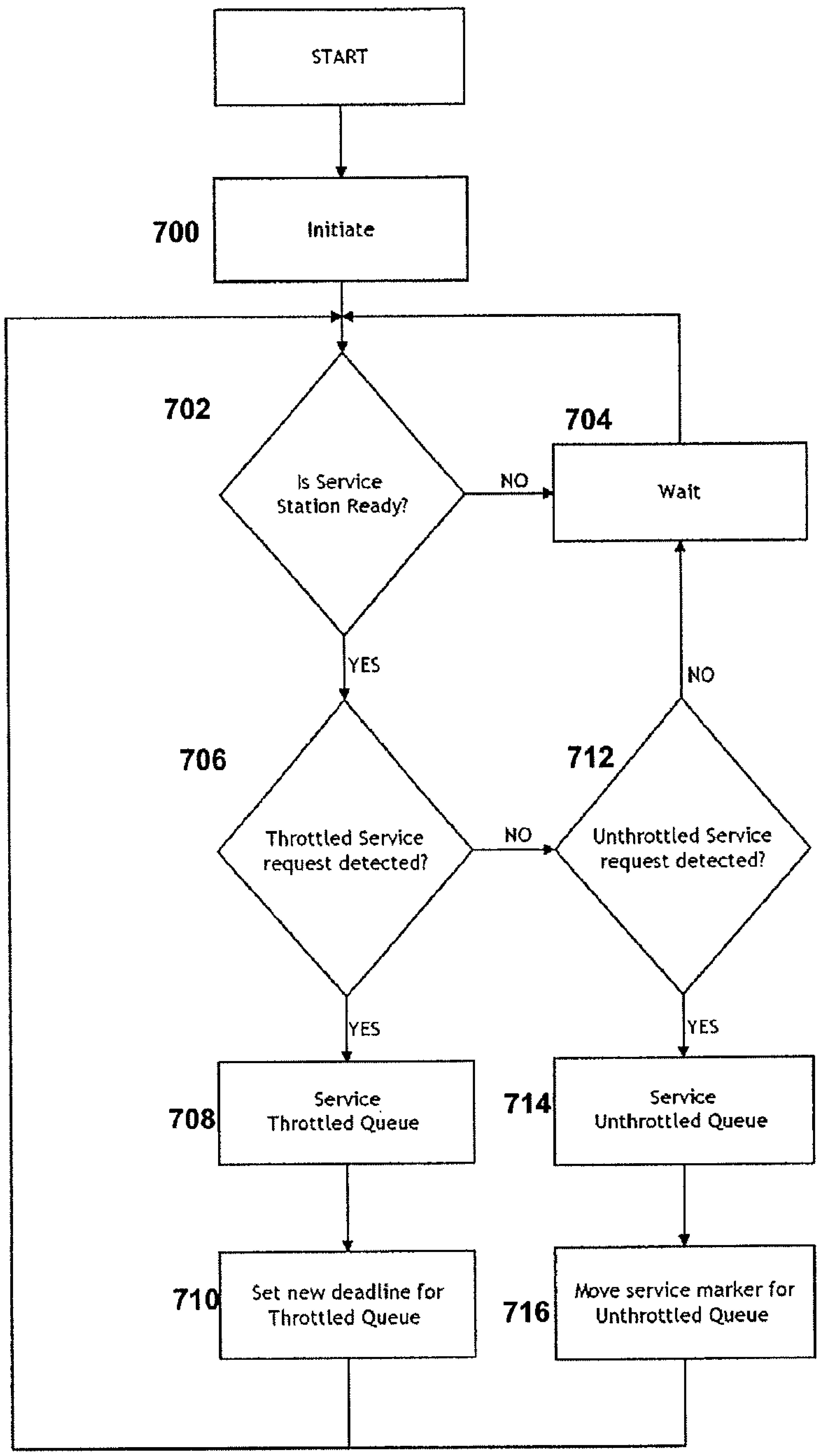

START
500
Initiate
Throttled Queues
502
Initiate deadlines for
Throttled Queues
504
Initiate
Gracefully Throttled Queues
506
Initiate service marker for
Gracefully Throttled Queues
508
Initiate
Unthrottled Queues
510
Initiate service marker for
Unthrottled Queues
512
Wait for
Service Request
514
Is
Service Request
Received?
NO
YES
516
Reorder
Service Request

Fig. 12



START
600
Initiate
602
Is Service Station Ready?
NO
604
Wait
YES
606
Throttled Service request detected?
NO
612
Gracefully Throttled Service request detected?
NO
YES
YES
608
Service Throttled Queue
614
Service Gracefully Throttled Queue
610
Set new deadline for Throttled Queue
616
Move service marker for Gracefully Throttled Queue

Fig. 13



START
700
Initiate
702
Is Service Station Ready?
NO
704
Wait
YES
706
Throttled Service request detected?
NO
712
Unthrottled Service request detected?
NO
YES
YES
708
Service Throttled Queue
714
Service Unthrottled Queue
710
Set new deadline for Throttled Queue
716
Move service marker for Unthrottled Queue

Fig. 14

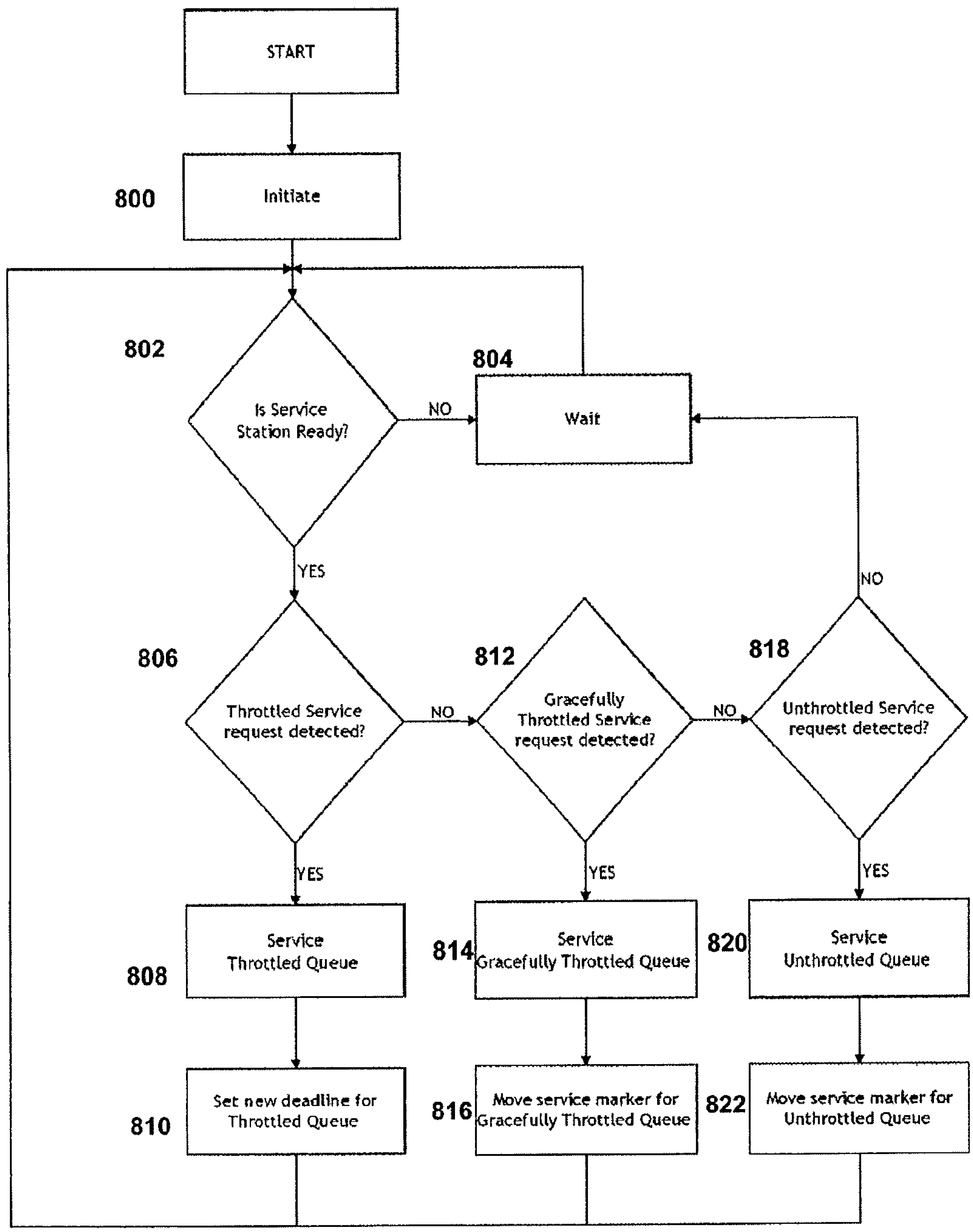

START
800
Initiate
802
Is Service Station Ready?
NO
804
Wait
YES
806
Throttled Service request detected?
NO
812
Gracefully Throttled Service request detected?
NO
818
Unthrottled Service request detected?
NO
YES
YES
YES
808
Service Throttled Queue
814
Service Gracefully Throttled Queue
820
Service Unthrottled Queue
810
Set new deadline for Throttled Queue
816
Move service marker for Gracefully Throttled Queue
822
Move service marker for Unthrottled Queue

Fig. 15

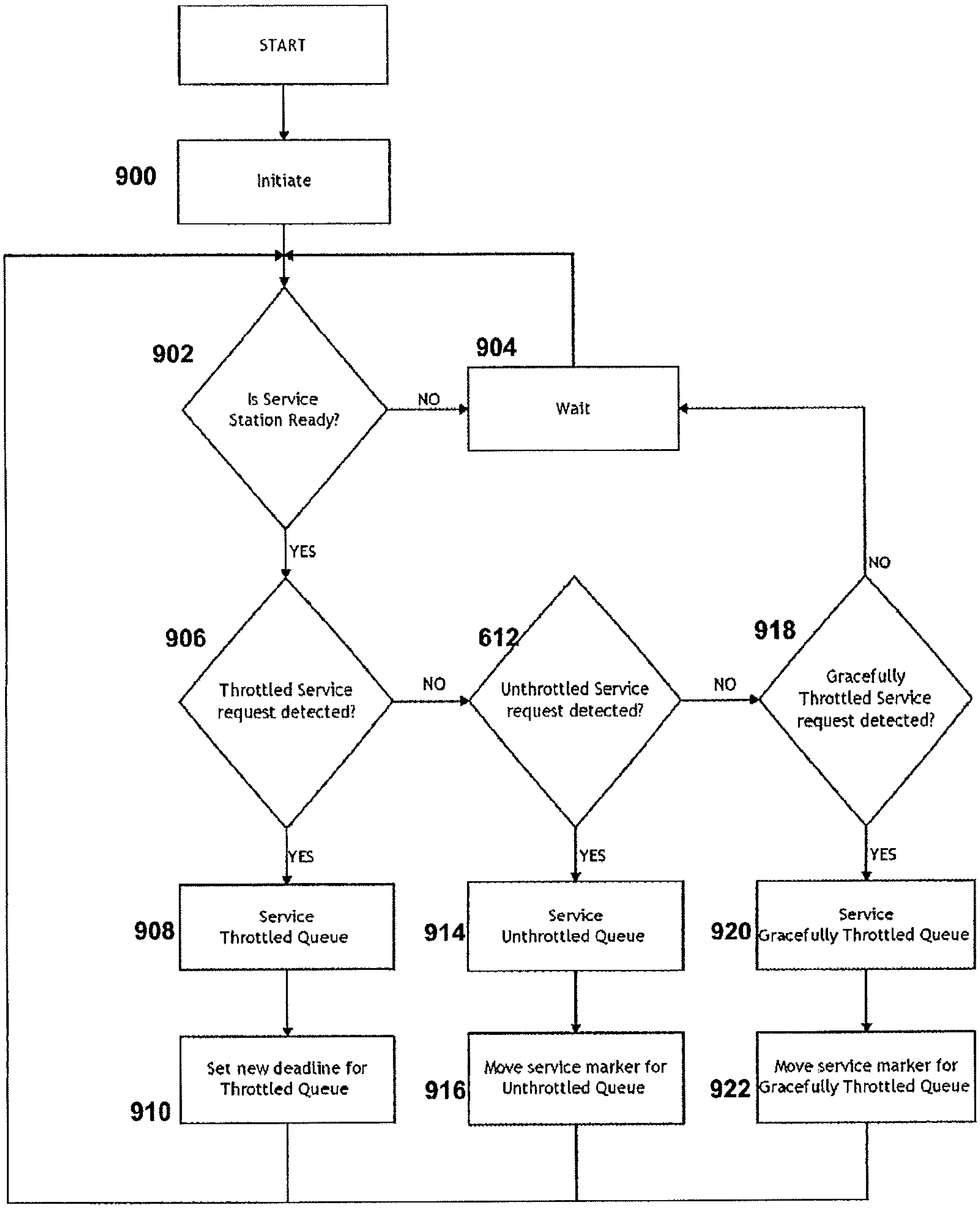
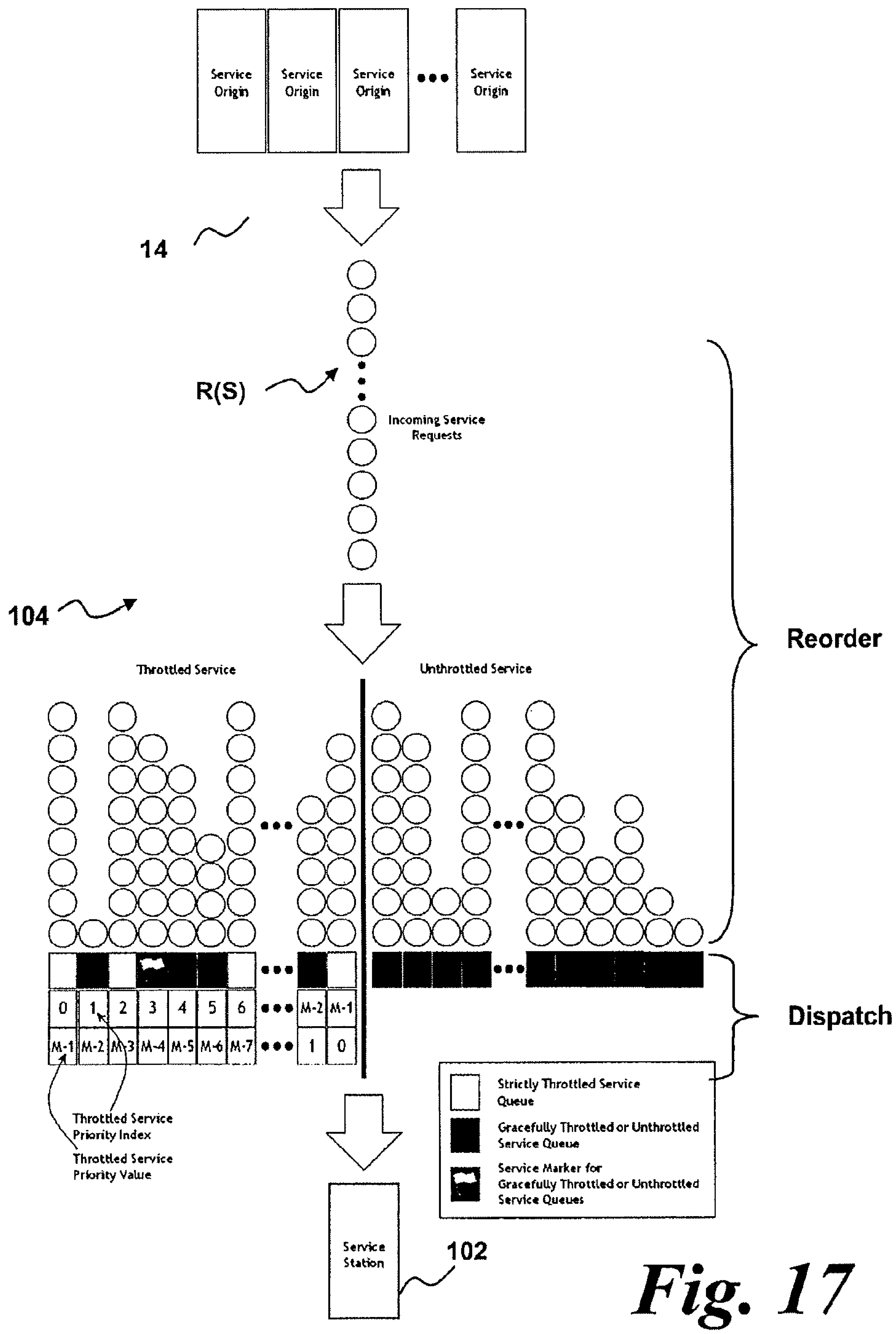

START
900
Initiate
902
Is Service Station Ready?
NO
904
Wait
YES
906
Throttled Service request detected?
NO
612
Unthrottled Service request detected?
NO
918
Gracefully Throttled Service request detected?
NO
YES
YES
YES
908
Service Throttled Queue
914
Service Unthrottled Queue
920
Service Gracefully Throttled Queue
910
Set new deadline for Throttled Queue
916
Move service marker for Unthrottled Queue
922
Move service marker for Gracefully Throttled Queue

Fig. 16



Service Origin
Service Origin
Service Origin
Service Origin
14
R(S)
Incoming Service Requests
104
Throttled Service
Unthrottled Service
Reorder
0
1
2
3
4
5
6
M-2
M-1
M-1
M-2
M-3
M-4
M-5
M-6
M-7
1
0
Dispatch
Throttled Service Priority Index
Throttled Service Priority Value
Strictly Throttled Service Queue
Gracefully Throttled or Unthrottled Service Queue
Service Marker for Gracefully Throttled or Unthrottled Service Queues
Service Station
102

Fig. 17

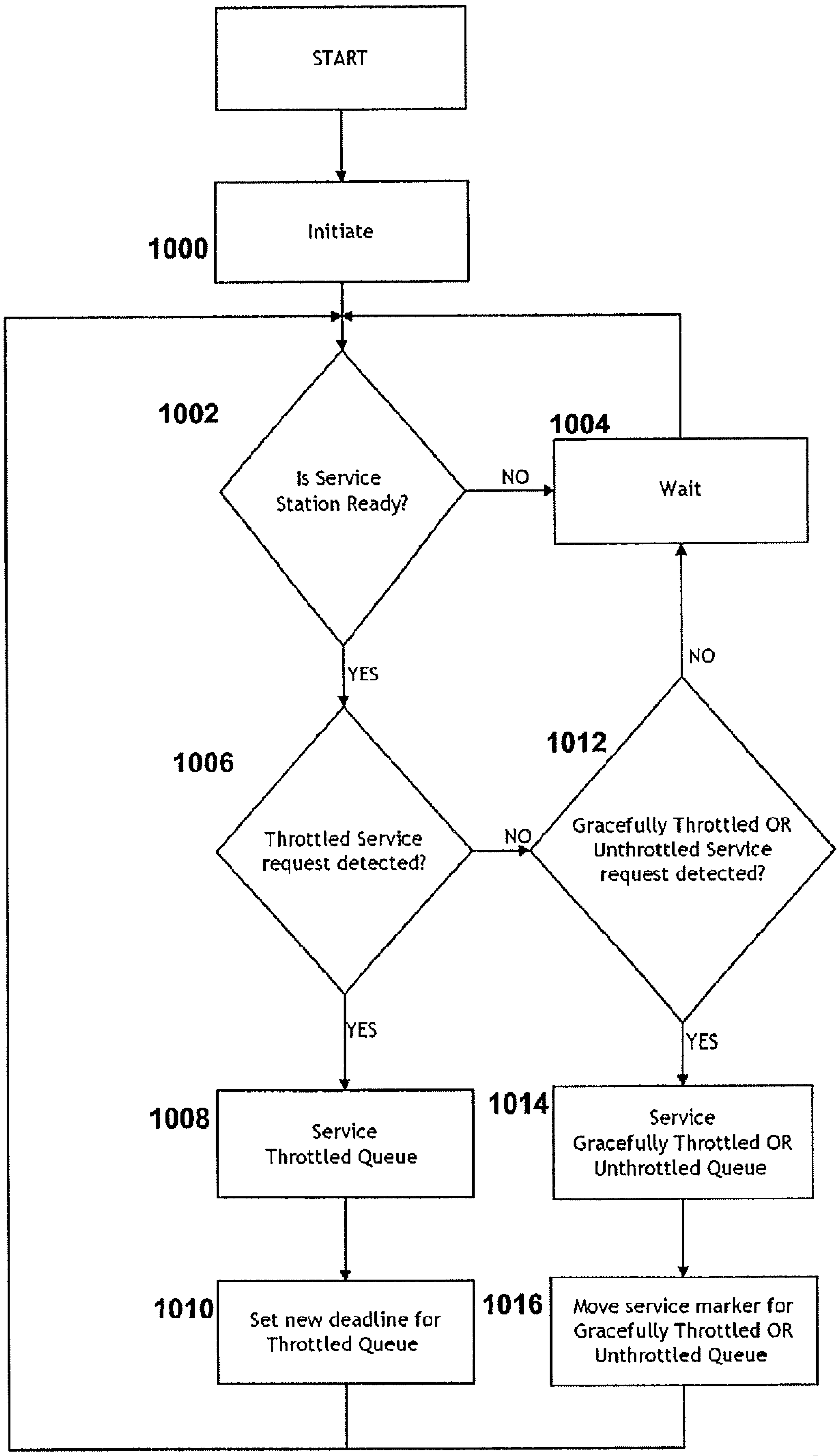

START
Initiate
1000
1002
Is Service Station Ready?
NO
1004
Wait
YES
NO
1006
Throttled Service request detected?
NO
1012
Gracefully Throttled OR Unthrottled Service request detected?
YES
YES
1008
Service Throttled Queue
1014
Service Gracefully Throttled OR Unthrottled Queue
1010
Set new deadline for Throttled Queue
1016
Move service marker for Gracefully Throttled OR Unthrottled Queue

Fig. 18

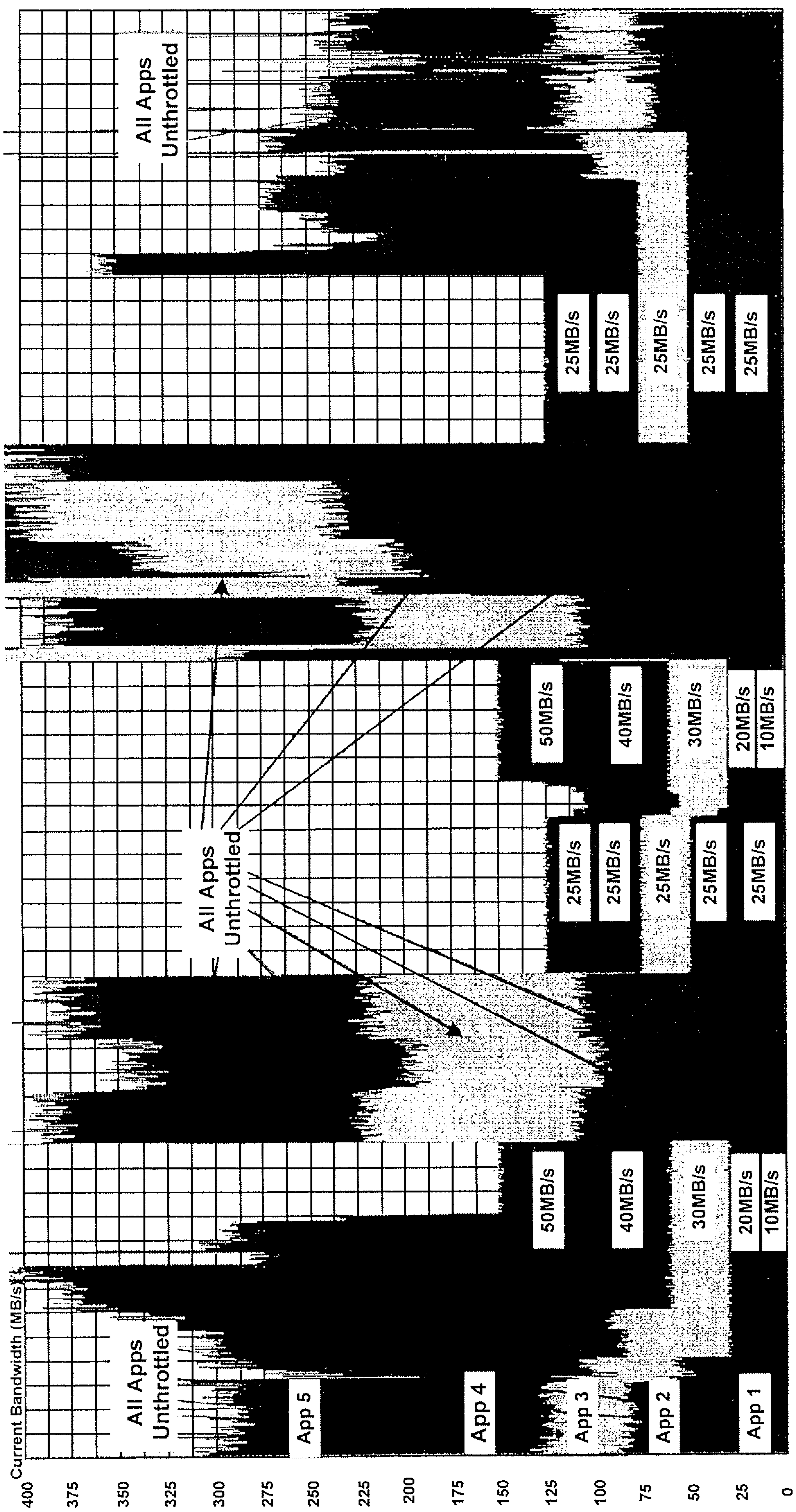

Current Bandwidth (MB/s)
400
375
350
325
300
275
250
225
200
175
150
125
100
75
50
25
0
All Apps Unthrottled
App 5
App 4
App 3
App 2
App 1
50MB/s
40MB/s
30MB/s
20MB/s
10MB/s
All Apps Unthrottled
25MB/s
25MB/s
25MB/s
25MB/s
25MB/s
50MB/s
40MB/s
30MB/s
20MB/s
10MB/s
25MB/s
25MB/s
25MB/s
25MB/s
25MB/s
All Apps Unthrottled

Fig. 19

SCHEDULING REQUESTS FOR DATA TRANSFERS IN A MULTI-DEVICE STORAGE SYSTEM

RELATED APPLICATION

This application is a continuation of copending U.S. patent application Ser. No. 13/193,005 filed on Jul. 28, 2011, now issued as U.S. Pat. No. 8,909,764.

BACKGROUND

The present disclosure relates to a method of, and apparatus for, scheduling requests for data communication from a client-side service to a service station. More particularly, the present disclosure relates to scheduling requests for data communication from a client-side service to a service station to enable provision of a guaranteed data rate for communication.

Traditionally, electronic data is stored locally on a user's computer system by means of a data storage resource such as a hard disk drive (HDD) or other storage media. However, the increasing prevalence of data-heavy resources (for example, real-time high definition video) has led to an increased demand for storage capacity.

An increasingly popular area is what is known as "cloud computing". Cloud computing provides a set of scalable and often virtual resources over a network such as an Ethernet or the Internet. A "cloud" comprises a consolidated storage system having large storage capacity (typically at the multi-petabyte level) which may serve independent customers (e.g. the cloud acts a storage service provider) or business units within an organisation (e.g. the cloud acts as a common corporate data store). In essence, cloud architecture means that the users generally do not own the physical computing resources they use and, instead, purchase usage from a third-party provider in a service-orientated architecture, or access a common corporate data store.

Cloud-type storage service providers are attractive to small to medium sized enterprises which do not typically have the resources to invest in over-provisioned storage infrastructures which will never be used efficiently. In addition, cloud-type services enable a user having multiple devices (e.g. smartphones, tablets, laptops and workstations) to access common stored data without the need to synchronise the data between the individual devices.

Storage service providers offer such users access to the storage services that they require without the need for capital expenditure on hardware and software solutions. In addition, the cost of hardware is becoming increasingly small in comparison to the cost of maintaining and managing a data storage resource. Therefore, this makes the cloud approach even more attractive to businesses. In many cases, service providers provide services in the manner of a utility service and billed, for example, on the basis of the storage resources (e.g. storage space) consumed by the user or on a periodical billing basis.

It is known for the provision of services by a service provider to be covered by service level agreements (SLAs). An SLA is a negotiated agreement between a service provider (or target) offering a service and a client (or initiator) requiring use of the service. The SLA records a common agreement regarding the quality of service (QoS) to be delivered to the client. For example, in the field of data storage provision, the QoS may relate to a particular level of storage capacity or reliability which can be guaranteed by the service provider.

Increasingly, users of a storage resource may wish to access bandwidth-intensive media such as streaming video data. In this regard, a minimum bandwidth is required to provide smooth playback of the streamed video. If the minimum bandwidth is not met or maintained by the storage resource, then there may be pauses in the video playback whilst the required data is obtained from the storage resource. This leads to an unsatisfactory user experience. As a result, some users of a storage resource may prefer to specify a minimum guaranteed bandwidth in addition to a guaranteed volume of storage space.

However, to date, it has been difficult to guarantee bandwidth in known storage resource arrangements. This is because the performance of a given data storage resource is heavily dependent upon the demands placed upon it. If a number of users are using a large proportion of bandwidth of the data storage resource, then the service provider may not be able to meet the particular bandwidth requirements specified by each user. Given the non-deterministic nature of storage resource access, this means that, currently, it is not possible to provide an assurance of a given bandwidth when the data is accessed.

Typically, the only way to circumvent this problem is to heavily over-provision the data storage resource, i.e. to have sufficient spare capacity to ensure that the specified bandwidth requirements are met. However, this approach is wasteful of resources and uneconomical because a significant proportion of the bandwidth available to the data storage resource must be kept free for use during abnormally heavy traffic conditions, and so is rarely used. Consequently, existing service-orientated storage providers can only guard against "worst case" scenarios of abnormally heavy load.

This issue can be mitigated by providing a "throttled" service. A throttled service is one which is bandwidth limited, as opposed to an "unthrottled" service which has no such bandwidth restriction and would, in the absence of competing data transfers, in principle consume all of the bandwidth available. Throttling of user's services may assist in preventing some users from consuming an unfair proportion of the available bandwidth. However, throttling in a general sense merely provides an upper limit on the bandwidth and cannot provide a minimum lower limit which is required in order to guarantee smooth transmission of, for example, video data as discussed above.

Therefore, known storage provision arrangements suffer from a technical problem that bandwidth requirements cannot be efficiently and accurately guaranteed. This means that real-time guarantees on storage resource bandwidth cannot be made without over-provisioning of the storage resource.

"*Argon:performance insulation for shared storage servers*" Wachs et al., 5th Usenix conference on file and storage technologies (FAST '07) and U.S. Pat. No. 7,917,903 relate to scheduling methods to enable a particular quality of service to be provided. However, these arrangements are unsuitable for parallel distributed file systems in which data is stored over a number of service systems, such as may be found in a cloud-type system.

SUMMARY

Various embodiments of the present disclosure are directed to an apparatus and method for scheduling requests for data transfers in a multi-device storage system.

In some embodiments, a system includes at least one server coupled to a pool of storage devices to transfer data from the storage devices to client devices responsive to requests. A request scheduler is adapted to receive into a memory a plurality of requests each having a service identifier (ID) and a payload size, to set a deadline for each request responsive to the service ID and the payload size, to forward the requests to the server for processing in an order based on service ID and, responsive to the deadline being reached for a selected request, to advance the selected request for immediate processing by the server.

In other embodiments, a system includes a storage server comprising a programmable processor and associated memory, a plurality of storage devices coupled to the storage server adapted to store data objects from client devices coupled to the storage server, a memory adapted to accumulated pending client requests for data objects from the client devices, and a request scheduler adapted to manage the pending client requests in the memory for processing by the storage server. The request scheduler has a programmable processor and associated memory adapted to establish a deadline for each pending client request responsive to a service ID value and a payload size value associated with the client request, to sort the pending client requests into a plurality of first-in-first out (FIFO) queues based on the service ID values, to forward the queued client requests from the FIFO queues to the storage server, to monitor the deadlines and to advance a selected queued client request for immediate execution by the storage server responsive to the associated deadline being met.

In other embodiments, a computer-implemented method includes storing data objects from client devices in a pool of data storage devices using at least one storage server; accumulating, in a memory, client requests to transfer the data objects to a number of client devices, each client request having a service ID value and a payload size value; determining a deadline for each client request responsive to the service ID value; forwarding the client requests to the at least one storage server for processing in an order based on the service ID values; and monitoring the deadlines and, responsive to the deadline being reached for a selected client request, advancing the selected client request for immediate processing by the at least one storage server.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the present disclosure will now be described in detail with reference to the accompanying drawings, in which:

FIG. **1** is a schematic diagram of a cloud network;

FIG. **2** is a schematic diagram of an embodiment of an electronic data store comprising a single service station;

FIG. **3** is a schematic diagram of an embodiment comprising a distributed environment comprising multiple service stations and multiple service origins;

FIG. **4** is a schematic diagram showing the communication between service origins, the request scheduler, request coordinator and service station;

FIG. **5** is a schematic diagram illustrating the distance parameter included with service requests;

FIG. **6** is a schematic diagram showing the processing of service requests in the request scheduler;

FIG. **7** is a flowchart illustrating the operation of the reordering process in the request scheduler;

FIG. **8** is a flowchart illustrating the operation of the dispatch process in the request scheduler;

FIG. **9** is a schematic illustration of a single FIFO queue as scheduled by the request scheduler;

FIG. **10** is a schematic illustration of a single FIFO queue as scheduled by the request scheduler;

FIG. **11** is a schematic diagram showing the processing of service requests in the request scheduler according to an alternative embodiment;

FIG. **12** is a flowchart illustrating the operation of the reordering process in the request scheduler according to an alternative embodiment;

FIG. **13** is a flowchart illustrating the operation of the dispatch process in the request scheduler according to an alternative embodiment;

FIG. **14** is a flowchart illustrating the operation of the dispatch process in the request scheduler according to an alternative embodiment;

FIG. **15** is a flowchart illustrating the operation of the dispatch process in the request scheduler according to an alternative embodiment;

FIG. **16** is a flowchart illustrating the operation of the dispatch process in the request scheduler according to an alternative embodiment;

FIG. **17** is a schematic diagram showing the processing of service requests in the request scheduler according to an alternative embodiment;

FIG. **18** is a flowchart illustrating the operation of the dispatch process in the request scheduler according to an alternative embodiment;

FIG. **19** is a schematic graph showing the operation of the present disclosure on a service station.

DETAILED DESCRIPTION

FIG. **1** shows a schematic illustration of an electronic data store **10** provided by a service provider. The data store **10** comprises a plurality of storage units **12**. Each storage unit **12** may take the form of, for example, an individual hard drive or a collection of hard disk drives (HDDs) linked together through a protocol such as Redundant Array of Inexpensive Disks (RAID) to form a logical unit. However, irrespective of the number or configuration of HDDs present, the data store **10** is presented to the service origins **14**-*i* as a single logical drive.

A plurality of service origins **14** connect to the data store **10** through a cloud network **16**. The service origins **14** may take any suitable form. For example, they may comprise workstations, mobile telephones (such as, for example, so-called "smartphones"), laptop computers, tablet computers, servers or other computing equipment. In the context of the described embodiments, the clients may take the form of one or more programs (for example, a web browser system) running on a computer hardware system. Additionally, the service origins **14** may comprise client programs which form part of a parallel file system, distributed file system or other distributed network services.

The cloud network **16** may take a number of forms, for example, an internet network, a cable network or a mobile network. The cloud network **16** enables each user of each client **14** to read data from, or write data to, the data store **10** as if the data was stored locally. Each client computer **14** has an SLA with the service provider of the data store **10** which specifies the QoS required by the user of the client computer **14** whilst connected to the data store **10**. For example, the SLA might specify the type of data access required (e.g. random or sequential) and/or the bandwidth/latency requirements of the access required to, or the retrieval required from, the data store **10**.

FIG. **2** shows a first embodiment of an electronic data store **100**. The electronic data store **100** comprises a service station **102** and a request scheduler **104**. In this example, the service station **102** comprises a server **106** and at least one data storage component **108**. In an example, the data storage component **108** may comprise a group of approximately five to eight physical drives linked together in a RAID arrangement.

RAID architecture combines a multiplicity of small, inexpensive disk drives into an array of disk drives that yields performance that can exceed that of a single large drive. This arrangement enables high speed access because different parts of a file can be read from different devices simultaneously, improving access speed and bandwidth.

Data interleaving in a RAID arrangement is usually in the form of data "striping" in which the data to be stored is broken down into blocks called "stripe units". The "stripe units" are then distributed across the physical drives. Therefore, should one of the physical drives in a group forming a storage component **108** fail or become corrupted, the missing data can be recreated from the data on the other drives. The data may be reconstructed through the use of the redundant "stripe units" stored on the remaining physical drives using known RAID techniques such as XOR.

The physical drives may take any form of storage device, such as, for example, tape drives, disk drives, non-volatile memory, or solid state devices. Although most RAID architectures use hard disk drives as the main storage devices, it will be clear to the person skilled in the art that the embodiments described herein apply to any type of suitable storage device. Further, a physical drive may take the form of a single partition on a hard disk drive. Therefore, a single hard disk drive may comprise a plurality of physical drives in the context of the electronic data store **100**.

In this embodiment, the data storage component comprises an Object-based Storage Device (OSD). An OSD is a computer storage device which operates at a higher level of abstraction than block-orientated interfaces. An OSD does not read and write fixed sized blocks of data as in a conventional file system structure. Instead, the OSD standard organizes data into variable-sized data packages known as objects. Each object is associated with data and metadata comprising an extensible set of attributes which describe the object.

However, the exact form and nature of the service station **102** is not material to the present disclosure. In the context of this application, the service station **102** is a single storage facility for data to which I/O requests can be communicated and to which data can be sent to, stored on, and retrieved from. The service station **102** may, in fact, comprise a number of different OSDs **108** connected together through a single server or through multiple servers. The skilled person would be readily aware of the different forms or structures that the service station **102** may take within the scope of the present disclosure.

The request scheduler **104** controls the flow of data into and out of the storage resource **102**, and controls access to the service station **102** from client computers **14** through a network **110** using, for example, the TCP/IP protocol. The network **110** may comprise a local area network (LAN) or the Internet using existing network infrastructure.

The request scheduler **104** may take a number of forms; for example, the request scheduler **104** may take the form of a metadata server. Alternatively, the request scheduler **104** may take the form of a software or hardware interface run on the service station **102**. The skilled person would be readily aware of the variations which fall within the scope of the present application.

The request scheduler **104** is operable to receive a plurality of service requests R from the service origins **14** and schedule the requests to provide a guaranteed service rate or service throughput i.e. a guaranteed rate of work done for groups of service requests across the entire distributed system where service requests with same Service IDs could originate at more than one Service Origins and reach more than one Service Stations for service completion. In general, the requests R comprise metadata identifying payload data (e.g. I/O read/writes) to be sent to the service stations from the service origins. By scheduling these requests, the flow of data to the service stations from the service origins can be scheduled and optimised to provide guarantees of bandwidth or other parameters as required.

The I/O requests R may be in any suitable format. For example, the I/O requests R may be in the OSD (Object storage Device) protocol. The OSD protocol uses a SCSI command set developed by the T10 committee of the International Committee for Information Technology Standards. In the OSD standard, objects are specified with a 64-bit partition ID and a 64-bit object ID. The command interface comprises storage commands to create and delete objects, to write bytes and read bytes to and from individual objects, and to "set" attributes on objects, and to "get" those attributes on objects. Alternatively, other formats for the requests R may be used, and these will be readily apparent to the skilled person.

An alternative embodiment is illustrated in FIG. **3**. This configuration illustrates a distributed networked environment **200**. This distributed environment **200** may represent any distributed environment where a number of service stations **102** exist to serve service requests R coming in from a number of service origins **14** where service requests R originate. The system may have one or more service coordinators **112** who helps service origins **14** send service requests R to service stations **102** and get their service completed.

The second embodiment of the data store **200** comprises a plurality of parallel but separate peer service stations **102**-*i*. Each service station **102**-*i* is substantially the same as for the previous embodiment and is connected to the network **110** whereby it is operable to receive requests R from one or more service origins **14**. Again, in this embodiment, the requests R comprise metadata identifying payload data (e.g. I/O read/writes) to be sent to the service stations from the service origins. However, service requests may be used for any suitable purpose and may not necessarily relate to I/O or read/write requests. The skilled person will be readily aware of variations which will fall within the scope of the present disclosure.

Whilst, in FIG. **3**, six service stations **102-1**, **102-2**, **102-3**, **102-4** and **102-5** are shown, in principle any number of service stations **102** may be provided as required.

Each service station **102**-*i* has a request scheduler **104**-*i*. The request scheduler **104**-*i* controls the organisation of requests into the service station **102**-*i*. The operational features of the request schedulers **104**-*i* may be implemented in either a hardware or software layer. The skilled person will be readily aware that the above features of the present embodiment could be implemented in a variety of suitable configurations and arrangements within the context of the present disclosure.

The service stations **102**-*i* communicate over the network **110** such as a local area network (LAN) or the Internet using existing network infrastructure, depending upon the relative location of the service stations **102**-*i* and the request scheduler **104**.

Each service coordinator **112** is also operable to communicate with one or more service stations **102**-*i* through the network **110**.

An example of the connections and communication between one or more service origins **14**, a service coordinator **112**, a request scheduler **104** and service station **102** is shown in FIG. **4**. The configuration shown may be used in either of the first or second embodiments.

As shown in FIG. **4**, each request R comprises a number of attributes. Depending upon the nature of the request R, these attributes may, for example, comprise I/O metadata at the head of an I/O request.

Each request R has a service ID S_x which identifies the respective service (e.g. software or hardware program or resource) which may be operating on one or more service origins **14**.

It is also envisaged that a service having service ID S_x (which may be, as previously described, a software application or other distributed service application or utility) may be distributed across a number of locations (e.g. across a number of service origins **14**, or different parts of single system, or merely distributed modules of a service).

Therefore, requests R may be received from a single service ID S_x may be received from multiple service locations. Consequently, a further parameter of location ID L_y is associated with each service ID S_x.

Therefore, as an example, a single service ID S1 may be spread across a plurality of locations having, for example, location IDs L1, L2 and L3. The location ID L_y is attached to each service request R.

Each request also has a request number n. The request number represents the number of the request R from service ID S_x and location ID L_y. Therefore, the n^{th} request from service ID S_x and location ID L_y would be represented as $R_n(S_x, L_y)$. However, whilst, for the purposes of this disclosure each request is associated with a request number n, this need not be the case and each request need not be numbered in practice.

Additionally, each request R has size metadata $w_n(S_x, L_y)$, i.e. metadata pertaining to the volume of data contained in the request $R_n(S_x, L_y)$ from service ID S_x and location ID L_y. The size $w_n(S_x, L_y)$ is known as the "weight" of the request R and determines, in part, how long the service station **102** will take to service the I/O request R_n.

Additionally, a further parameter of distance $x_n(S_x, L_y)$ is attached with each request. Since, with plural service stations **102-1** to **102-*i*** receiving requests $R_n(S_x, L_y)$ from a plurality of different locations and service IDs, a number of different requests from a particular service ID and location ID may be handled in parallel by different service stations **102-*i***. Consequently, in order to provide a deadline system which, on aggregate, meets the required service throughput of the service ID S_x for each location ID L_y, the request $R_n(S_x, L_y)$ must comprise sufficient information to set a deadline without a priori knowledge of the processing being carried out in other service stations **102**.

In other words, each request $R_n(S_x, L_y)$ must comprise self-contained information regarding the relationship of that particular request to previous requests such that request $R_n(S_x, L_y)$ can be processed in the correct timeframe as will be described.

In a sequence of incoming requests for a particular service ID S having location ID L, the distance $x_n(S_x, L_y)$ for the n^{th} request is calculated from equation (1) below:

$$x_n(S_x, L_y) = \sum_{i=0}^{n} w_i(S_x, L_y) \tag{1}$$

where x_n is the distance for request n, and w_i is the weight (I/O size) for request i where i is in the range of 0 to n.

Take, for example, a service ID S1 having location ID L1 and, therefore, requests $R_n(S1, L1)$. In this case, the distance x_n for request n for service ID S1 at location L1 comprises the sum of the I/O sizes of requests 0 (i.e. the first request) to request n−1 from service ID S1 at location ID L1, plus the weight w_n of request $R_n(S1, L1)$ itself.

Concomitantly, for location ID L2 of service ID S1, the distance x_n for request n for service ID S1 at location L2 comprises the sum of the I/O sizes of requests 0 (i.e. the first request) to request n−1 from service ID S2 at location ID L2, plus the weight w_n of request $R_n(S1, L2)$ itself.

The concept of distance is shown diagrammatically in FIG. **5** where the distance parameter can be visualised as the sum of the weights of previous requests plus that of the current request.

In summary, a deadline for a particular request can be utilised which is dependent upon the previous requests from a particular service ID S_x and for a particular location ID L_y, i.e. the "distance" from the earlier requests from that service ID S_x and that location ID L_y. In this way, an aggregated minimum bandwidth can be guaranteed without administration from a central server attempting to handle distribution across a plurality of service stations to meet a guaranteed bandwidth.

Should a minimum bandwidth be guaranteed for a particular service having service ID S_x, service throughput metadata $T(S_x)$ is provided for service ID S_x. In this embodiment, the service throughput metadata $T(S_x)$ is supplied by the service coordinator **112** to the request scheduler **104**. However, alternatives are possible and the request scheduler **104** may, for example, contain an internal record of the service throughput metadata $T(S_x)$.

The service throughput metadata $T(S_x)$ specifies, for service ID S_x, the minimum bandwidth that the service S_x requires. This information is supplied to the request scheduler **104** and is associated with a particular service ID S_x. The service throughput information may, for example, be supplied to the request scheduler **104** in the form of an SLA previously agreed prior between the client or service prior to request transmission.

As will be described, the request scheduler **104** is further operable to process and schedule the requests $R_n(S_xL_y)$ to determine the appropriate order in which to service the requests. The request scheduler **104** is then operable to send the requests $R_n(S_x, L_y)$ in the appropriate order for servicing by the service station **102**.

To illustrate how the parameters of each request $R_n(S_x, L_y)$ are used to provide a guarantee of bandwidth, the method of operation of the present disclosure will now be described with reference to FIGS. **6** to **8**. FIG. **6** shows a schematic diagram of the movement of a request $R_n(S_x, L_y)$ from the service origin **14** to the service station **102**. FIGS. **7** and **8** show flow charts representing the method of an embodiment of the disclosure.

The request scheduler **104** has two independent stages of operation. The first stage is to reorder incoming requests. Once the requests have been reordered, then the second stage of dispatching the requests to the service station **102** can be carried out.

The description of the reordering stage is outlined below.

Step **300**: Initiate Throttled Queues

At step **300**, the request scheduler **104** conFIG.s a plurality of First-In First-Out (FIFO) queues for service.

Step **302**: Initiate Deadlines for Throttled Queues

At step **302**, an initial service deadline $d_n(S)$ is set. The service deadline $d_n(S)$ specifies the end of a time period within which request n in a queue of requests having service

ID S must be serviced in order to meet the service throughput requirements of the throttled queue for a particular service ID S. Initially, when requests are first received, they can effectively be served immediately. Therefore, the initial deadline is set to the current time.

Step **304**: Wait for Service Request

At step **304**, the request scheduler **104** waits for a service request to be received from one of the service origins. The method then proceeds to step **306**.

Step **306**: Service Request Received?

At step **306**, it is determined whether a service request has been received. If no service request has been received, the method proceeds back to step **306**. However, if a service request is received, the method then proceeds to step **310**.

The request scheduler **104** receives the I/O requests $R_n(S_x)$, each of which includes the service ID S and I/O size $w_n(S_x)$ and, optionally, may have service throughput metadata $T(S_x)$ associated therewith, from the service origin **14** via the network.

Step **308**: Reorder Incoming Requests by Service ID

Once the requests $R_n(S_x, L_y)$ have been received, the request scheduler **104** reorders the requests into First-In First-Out (FIFO) queues for service. At step **308**, the incoming requests $R_n(S_x, L_y)$ are sorted by service ID S_x and allocated to an individual service so that each queue contains only requests $R_n(S_x, L_y)$ from the particular service having that service ID S_x. The method then proceeds back to step **304** to process and reorder other incoming requests.

The method above operates continuously in the request scheduler **104** when in operation. Once the requests are organised into appropriate queues, then they can be dispatched to the respective service station **102** in the next stage of operation.

The dispatch process will be described with reference to FIG. **8**.

Step **400**: Initiate

At step **400**, the dispatch process is initiated. The method then proceeds to step **402**.

Step **402**: Is Service Station Ready?

At step **402**, it is determined whether the service station is ready to receive a new service request, i.e. if it has finished servicing the previous request the service station **102** has been handling. If not, the method proceeds to step **404**.

If the service station is ready to service a new request, the method proceeds to step **406**.

Step **404**: Wait

At step **404**, the method waits for a predetermined time to enable the service station to complete the current request and be ready to receive a new request. This time period may be any suitable time period.

Step **406**: Throttled Service Request Detected?

The request scheduler **104** examines the service deadline $d_n(S_x, L_y)$ for each throttled queue, starting with the highest priority queue (in this example, the leftmost queue in FIG. **3**), i.e., where there are M services, the queue having the highest priority value (M−1). Consequently, the $M-1^{th}$ service has a priority value of 0, i.e. the lowest value of priority, and this queue will be served last in the throttled queues. If the deadline for the highest priority throttled queue has not yet been reached, the request scheduler **104** moves to the next throttled queue in line in priority order.

Priority ordering provides predictable performance degradation or performance improvement when system capability becomes respectively low or high. The lower priority services will not meet deadlines and will suffer from performance degradation when the system capability becomes low and not enough to server all throttled queues. The lower priority services will improve gradually as system capability improves.

Additionally, the throttling mechanism falls back to a priority based servicing when the system capability is not enough to meet all throttled service requests. In other words, setting high throughput values for all or some of the queues means deadlines will expire quickly and queues will be served in their priority order.

This process continues until a throttled queue is reached which has a deadline which has been reached or has passed. At this point, the method proceeds to step **408**.

If, however, the request scheduler **104** reaches the end of the throttled queues and none of the queues has a deadline which has passed, then the method proceeds back to step **404**.

Step **408**: Serve Queue for which Deadline has been Reached or Passed

If, in step **406**, it is detected that a queue has a deadline which has been reached or has expired, then the request scheduler **104** will service first request $R_n(S_x, L_y)$ in line in that queue. Since the examining step **406** is carried out in priority order, in practice this means that requests $R_n(S)$ in the throttled queues are serviced starting with the throttled queue having the highest priority for which the deadline $d_n(S_x, L_y)$ for servicing has been reached or has passed.

When a queue is serviced, the request scheduler **104** passes the respective I/O request $R_n(S_x, L_y)$ at the head of that particular queue to the service station **102** for processing.

Step **410**: Set New Deadline for Serviced Queue

At step **410**, once the throttled queue has been serviced in step **408**, a new deadline is set by which time the next request $R_n(S_x, L_y)$ in the FIFO queue is to be serviced. The new deadline $d_n(S_x, L_y)$ to be set is dependent upon the nature of the system and the required throughput.

The simplest case is for the configuration shown in FIG. **2**. In this arrangement, a service having service ID S_x originating from a single location having location ID L_y sends requests to a single service station **102**.

In this particular case, the location ID L_y is not needed. Therefore, it may optionally be omitted from the service request R. Alternatively, if the location ID L_y is included in the service request R, it can be effectively ignored by the service station **102** because there will only be one location ID L_y in each queue.

Additionally, where a single service station **102** is used, the distance parameter is not required. Therefore, the service request R in this situation may omit the distance parameter. Alternatively, the distance parameter may simply be ignored by the service station **102**.

Consequently, requests from a single service ID at a single location to a single service station need only comprise the parameters of service ID and weight. Other parameters may be required and not used, depending upon the particular network configuration.

For the configuration of a single service sending requests from a single location to a single service station, the deadline $d_n(S_x)$ (where y is a constant) is then set based on the I/O size $w_n(S_x)$ of, and service throughput metadata $T(S_x)$ associated with, the next throttled request $R_n(S_x)$ in line in the relevant queue. Therefore, the deadline is set according to equation (2):

$$d_n(S_x) = c.t. + \frac{w_n(S_x)}{T(S_x)} \tag{2}$$

where d_n is the deadline for the queue for requests having service ID S_x, c.t. is the current time, w_n is the I/O size (in bytes) and $T(S_x)$ is the required service throughput (in Mb/s) associated with service ID S_x.

In other words, the new deadline is the current time plus the time required to process a given amount of data in a request at a required data rate (or bandwidth). Meeting this deadline will mean that the required service throughput is achieved.

In an alternative configuration, a service having service ID S_x and a single location ID L_y may send requests to multiple service stations. In this arrangement, each service station **102** sets deadlines independently of each other service station **102** and there is no communication therebetween. Consequently, it is necessary to utilise the distance parameter. However, again, in this example the location ID L_y is effectively a constant, and so it can be ignored in the following description.

The new deadline $d_n(S_x)$ for the next request $R_n(S_x)$ set in dependence upon a number of factors. As for the first example, the deadline $d_n(S_x)$ for request $R_n(S_x)$ is dependent upon the I/O size $w_n(S_x)$ and service throughput metadata $T(S_x)$ for the requests having service ID S_x. However, the deadline is set with the aim of achieving aggregated throughput across all of the service stations **102-1-102-***i*, i.e. achieving the requirements set out in the service throughput metadata $T(S_x)$ but in parallel with other service stations **102** processing requests from the same service having service ID S. Therefore, the distance parameter x is required, together with a further parameter, the last served distance l.

The last served distance parameter l is, effectively, the distance of the last served request in the queue for service ID S_x. In other words, for request $R_n(S_x)$ the last served distance l_n would be equal to the distance of the previously served request $R_m(S_x)$ (i.e. the request most recently processed in step **408**), as set out in equation (3):

$$l_n(S_x) = \sum_{i=0}^{m} w_i(S_x) \tag{3}$$

where l_n is the last served distance for request $R_n(S_x)$ which is equal to the distance of request $R_m(S_x)$, i.e. sum of the weights of requests $R_0(S_x)$ to $R_m(S_x)$.

The deadline for request $R_n(S_x)$ is, thus, calculated according to equation (4):

$$d_n(S_x) = c.t. + \frac{(x_n(S_x) - l_n(S_x))}{T(S_x)} \tag{4}$$

where d_n is the deadline for request $R_n(S_x)$, c.t. is the current time, w_n is the I/O size (or weight) (in bytes) of request R_n, x_n is the distance parameter of request n, l_n is the last served distance for request R_n (equal to the distance of previously-served request R_m) of service station **102-***j* and T is the required service throughput (in Mb/s).

In other words, the new deadline is the current time plus the time required to process a given amount of data in a request at a required data rate (or bandwidth) in order to achieve an aggregated throughput across all of the service stations **102-***i* to which requests having a particular service ID S have been sent. Meeting this deadline will mean that the required service throughput is achieved.

FIG. **9** provides an example of this. In a queue comprising service requests from service ID S1 having a single location ID L1 (omitted from the FIG. for clarity), the head request $R_3(S1)$ is immediately followed by request $R_5(S1)$.

$R_3(S1)$ is the first request in the queue, and so has a distance x_3 equal to $w_1+w_2+w_3$, and a last served distance l_3 equal to 0 (because this is the first request in the queue, there is no last served distance). Therefore, the deadline $d_3(S1)$ for request $R_3(S1)$ is equal to equation (5):

$$d_3(S1) = c.t. + \frac{(w1 + w2 + w3) - 0}{T(S1)} \tag{3}$$

Once request $R_3(S1)$ has been served, $R_5(S1)$ is the next request in line for servicing. The distance x_5 for $R_5(S1)$ is equal to $w_1+w_2+w_3+w_4+w_5$ and the last served distance l_5 for $R_5(S1)$ is equal to the distance of $R_3(S1)$ or $w_1+w_2+w_3$. Therefore, the deadline for $R_5(S1)$ is equal to equation (6):

$$d_5(S1) = c.t. + \frac{(w1 + w2 + w3 + w4 + w5) - (w1 + w2 + w3)}{T(S1)} \tag{6}$$

Which means that the new deadline for request $R_5(S1)$ is equal to equation (7):

$$d_5(S1) = c.t. + \frac{w4 + w5}{T(S1)} \tag{7}$$

In the final case, services having a service ID S_x originate from a number of locations having location IDs L_y and are serviced by a number of service stations **102**. This is done by utilising a deadline for a particular request which is dependent upon the previous requests from a particular service ID S_x and for a particular location ID L_y, i.e. the "distance" from the earlier requests from that service ID S_x and for that location ID L_y. In this way, an aggregated minimum bandwidth can be guaranteed without administration from a central server attempting to handle distribution across a plurality of service stations to meet a guaranteed bandwidth.

In a sequence of incoming requests for a particular service ID S having location ID L, the distance $x_n(S_x, L_y)$ for the n^{th} request is calculated from equation (8) below (which is identical to equation (1)):

$$x_n(S_x, L_y) = \sum_{i=0}^{n} w_i(S_x, L_y) \tag{8}$$

where x_n is the distance for request n, and w_i is the weight (I/O size) for request i where i is in the range of 0 to n.

Take, for example, a service ID S1 having location ID L1 and, therefore, requests $R_n(S1, L1)$. In this case, the distance x_n for request n for service ID S1 at location L1 comprises the sum of the I/O sizes of requests 0 (i.e. the first request) to request n−1 from service ID S1 at location ID L1, plus the weight of request $R_n(S1, L1)$ itself.

Concomitantly, for location ID L2 of service ID S1, the distance x_n for request n for service ID S1 at location L2 comprises the sum of the I/O sizes of requests 0 (i.e. the first request) to request n−1 from service ID S2 at location ID L2, plus the weight w_n of request $R_n(S1, L2)$ itself.

Knowing the time in which these requests need to be serviced (i.e. the I/O sizes divided by the service throughput

T(S1)) enables a deadline system which is able, on aggregate, to meet bandwidth requirements across the distributed system and to meet these requirements fairly across all of the locations of the service having service ID S1. Where l_m is the last served distance for request $R_n(S_x, L_y)$ which is equal to the sum of the weights of requests $R_0(S_x, L_y)$ to $R_m(S_x, L_y)$ as set out in equation (9):

$$l_n(S_x, L_y) = \sum_{i=0}^{m} w_i(S_x, L_y) \tag{9}$$

The deadline for request $R_n(S, L)$ is, thus, calculated according to equation (10):

$$d_n(S_x, L_y) = c.t. + \frac{(x_n(S_x, L_y) - l_n(S_x, L_y))}{T(S_x, L_y)} \tag{10}$$

where d_n is the deadline for request $R_n(S_x, L_y)$, c.t. is the current time, w_n is the I/O size (or weight) (in bytes) of request $R_n(S_x, L_y)$ x_n is the distance parameter of request n, l_n is the last served distance for request $R_n(S_x, L_y)$ (equal to the distance of previously-served request $R_m(S_x, L_y)$ of service station **102** and $T(S_x)$ is the required service throughput (in Mb/s).

Therefore, in this embodiment, the deadlines for each service ID S_x having location ID L_y are effectively independent between location IDs. In other words, within a single queue, the deadline for request $R_n(S1, L1)$ is dependent upon the distance of the most recently-served request $R_m(S1, L1)$ having the same location ID L1. Other requests in the same queue having different location IDs do not affect the deadline for request $R_n(S1, L1)$.

Therefore, in this embodiment, the last served distance is not necessarily that of the previous request in the queue for service ID S_x, but for the previous request in the queue having the same location ID L_y.

In other words, for a given location ID L_y, the new deadline is the current time plus the time required to process a given amount of data in a request at a required data rate (or bandwidth) in order to achieve an aggregated throughput across all of the service stations **302** to which requests having a particular service ID S_x have been sent. Meeting this deadline will mean that the required service throughput is achieved.

FIG. **10** provides an example of this. In a queue comprising service requests from service ID S1 and location IDs L1 to L3, the head request $R_3(S1, L1)$ is immediately followed by the following sequence of requests: $R_2(S1, L2)$, $R_3(S1, L3)$, $R_5(S1, L1)$, $R_{10}(S1, L2)$ and $R_{100}(S1, L3)$.

$R_3(S1, L1)$ is the first request in the queue, and so has a distance $x_3(S1, L1)$ equal to $w_1(S1, L1)+w_2(S1, L1)+w_3(S1, L1)$ and a last served distance $l_3(S1, L1)$ equal to 0 (because this is the first request in the queue, there is no last served distance). Therefore, the deadline $d_3(S1, L1)$ for request $R_3(S1, L1)$ is equal to equation (5), as shown in equation (11):

$$d_3(S1, L1) = c.t. + \frac{(w1(S1, L1) + w2(S1, L1) + w3(S1, L1)) - 0}{T(S1)} \tag{11}$$

Once request $R_3(S1, L1)$ has been served, $R_2(S1, L2)$ is the next request in line for servicing. The distance $x_2(S1, L2)$ for $R_2(S1, L2)$ is equal to $w_1(S1, L2)+w_2(S1, L2)$ and the last served distance l_2 for $R_2(S1, L2)$ is equal to 0, since this is the first request for location ID L2. The deadline $d_2(S1, L2)$ can then be calculated as for $R_3(S1, L1)$ in equation 11) above.

Once request $R_2(S1, L2)$ has been served, the next request in line is $R_3(S1, L3)$. The distance $x_3(S1, L3)$ for $R_3(S1, L3)$ is equal to $w_1(S1, L3)+w_2(S1, L3)+w_3(S1, L3)$ and the last served distance $l_3(S1, L3)$ for $R_3(S1, L3)$ is equal to 0, since this is the first request for location ID L3. The deadline $d_3(S1, L3)$ can then be calculated as for $R_3(S1, L1)$ in equation 11) above.

The next request in line is $R_5(S1, L1)$. The distance $x_5(S1, L1)$ for $R_5(S1, L1)$ is equal to $w_1(S1, L1)+w_2(S1, L1)+w_3(S1, L1)+w_4(S1, L1)+w_5(S1, L1)$ and the last served distance l_5 for $R_5(S1, L2)$ is equal to the distance of the previously served request for location ID L1, namely $R_3(S1, L1)$, or $w_1(S1, L1)+w_2(S1, L1)+w_3(S1, L1)$.

Therefore, the deadline $d_5(S1, L1)$ is equal to equation (12):

$$d_5(S1, L1) = c.t. + \frac{(w1(S1, L1) + w2(S1, L1) + w3(S1, L1)w4(S1, L1) + w5(S1, L1)) - (w1(S1, L1) + w2(S1, L1) + w3(S1, L1))}{T(S1)} \tag{12}$$

which means that the new deadline for request $R_5(S1)$ is equal to:

$$d_5(S1, L1) = c.t. + \frac{w4(S1, L1) + w5(S1, L1)}{T(S1)} \tag{13}$$

The penultimate and final requests in the queue shown in FIG. **10**, $R_{10}(S1, L2)$ and $R_{100}(S1, L3)$, can be calculated in a similar manner with, for example, $R_{100}(S1, L3)$ having a last served distance, $l_{100}(S1, L3)$ equal to the distance of the previously served request from location ID L3, namely $R_3(S1, L3)$.

Once the new deadline has been set, the method then proceeds back to step **402**.

It will be appreciated that not all of the above variables are necessary in each case. For example, as set out in the case of a single service station arranged serve all requests in a distributed system, the distance parameter is not required. Since all service requests with weights are directed toward a single service station the service station itself can determine the distance if necessary.

Additionally, some service requests may come with zero weight meaning that the service station that receives this service request wastes no efforts to complete this service request.

In the above disclosure, the service rate, i.e., the rate at which a service station works to complete a service request is Weight/Time where Time is time taken to complete service request.

For a group of service requests, i.e., service requests with same service ID, service throughput corresponds to average service rate over all the service requests completed so far.

The above embodiments provide configurations where bandwidth can be guaranteed for throttled services in, in one example, a distributed environment. However, throttled services alone may not fully utilise the capacity of a service station. In order to compensate for this, the following embodiments provide for additional services to use any remaining capacity without compromising the bandwidth guarantees for the throttled services. The two additional services that may be provided comprise gracefully throttled services and unthrottled services.

Gracefully throttled services are services which have a guaranteed minimum bandwidth but which are not restricted to that minimum limit and may consume more bandwidth if the capacity is available to do so. If the service station has capability to do so after servicing all the throttled service requests, the service station may increase the service throughput for particular service IDs. Gracefully throttled services are identified by metadata received by the service station identifying particular service IDs as relating to a gracefully throttled service. This metadata may be sent as part of the service throughput metadata, or may comprise separate metadata.

Unthrottled services are services which have no minimum bandwidth requirements at all. The service station may accommodate these service requests if there is capability to do so. Unthrottled requests correspond to service IDs that have no associated service throughput requirements.

The scheduler gives first priority to throttled service requests thereby guaranteeing the service throughput of service IDs with throughput requirements. After this, the gracefully throttled and unthrottled service requests can be serviced using a number of fair scheduling mechanisms. An example of this may be a "round-robin" scheduling.

FIG. **11** shows a schematic (similar to FIG. **6**) showing the general process of operation. In this embodiment, the queues are divided into throttled services (shown in white) and unthrottled services (shown in grey). Gracefully throttled services are also shown in black in the throttled section. FIG. **12** shows a flow chart of the reordering process of this embodiment.

Step **500**: Initiate Throttled Queues

At step **500**, the request scheduler **104** conFIG.s a plurality of First-In First-Out (FIFO) queues for service.

Step **502**: Initiate Deadlines for Throttled Queues

At step **502**, an initial service deadline $d_n(S_x)$ is set. The service deadline $d_n(S_x)$ specifies the end of a time period within which request n in a queue of requests having service ID S_x must be serviced in order to meet the service throughput requirements of the throttled queue for a particular service ID S_x. Initially, when requests are first received, they can effectively be served immediately. Therefore, the initial deadline is set to the current time.

Step **504**: Initiate Gracefully Throttled Queues

At step **504**, the request scheduler **104** conFIG.s a plurality of First-In First-Out (FIFO) gracefully throttled queues for service.

Step **506**: Place Service Marker for Gracefully Throttled Queues

At step **506**, a service marker (shown in FIG. **11**) is placed next to the next gracefully throttled service queue in line for servicing. The gracefully throttled queue to which the service marker is placed may be selected on any appropriate basis; for example, by queue priority, by a round-robin system or by a random selection.

Step **508**: Initiate Unthrottled Queues

At step **508**, the request scheduler **104** conFIG.s a plurality of First-In First-Out (FIFO) queues for service as unthrottled queues.

Step **510**: Place Service Marker for Unthrottled Queues

At step **510**, a service marker is placed next to the first unthrottled service queue to demark the throttled and unthrottled service queues.

Step **512**: Wait for Service Request

At step **512**, the request scheduler **104** waits for a service request to be received from one of the services. The method then proceeds to step **514**.

Step **514**: Service Request Received?

At step **514**, it is determined whether a service request has been received. If no service request has been received, the method proceeds back to step **512**.

However, if a service request is received, the method then proceeds to step **516**.

The request scheduler **104** receives the I/O requests $R_n(S_x)$, each of which includes the service ID S_x and I/O size $w_n(S_x)$ and, optionally, may have service throughput metadata $T(S_x)$ associated therewith, from the service origin **14** via the network. The I/O requests $R_n(S_x)$ comprise metadata relating to a data payload from the service having service ID S_x.

Step **516**: Reorder Incoming Requests by Service ID and Type

Once the requests $R_n(S_x, L_y)$ have been received, the request scheduler **104** reorders the requests into First-In First-Out (FIFO) queues for service. At step **518**, the incoming requests $R_n(S_x, L_y)$ are sorted by service ID S_x and allocated to an individual service so that each queue contains only requests $R_n(S_x, L_y)$ from the particular service having that service ID S_x.

As shown in FIG. **11**, the requests $R_n(S_x)$ are ordered into throttled and unthrottled service queues. Requests $R_n(S_x)$ for the throttled queues are identified by the presence of service throughput metadata $T(S_x)$ relating to the specific service ID S_x of the request R. The service throughput metadata $T(S_x)$ identifies that particular request $R_n(S_x)$ as coming from a client or service which has specified a particular minimum bandwidth requirement.

Requests $R_n(S_x)$ in the unthrottled queues are requests that will be addressed if sufficient capacity on the service station **102** exists to do this once the requests $R_n(S_x)$ in the throttled queues have been addressed. Unthrottled requests $R_n(S_x)$ do not have any service throughput metadata $T(S_x)$ associated therewith. The bandwidth remaining after the throttled service requests have been serviced will be shared between the unthrottled requests in the unthrottled request queues.

There is also a priority order to the throttled queues. As shown in FIG. **12**, in a given service cycle the leftmost queue (having the highest priority) will be serviced first, followed by queues to the right. Finally, once the throttled queues have been served, the unthrottled queues will be served if sufficient bandwidth remains.

The priority order may be set in a number of ways. An arbitrary system would be for the first users to purchase or access the storage to be allocated the highest priority. Another scheme would be for users to purchase a particular priority. The method then proceeds back to step **512** to process and reorder other incoming requests.

The method above operates continuously in the request scheduler **104** when in operation. Once the requests are organised into appropriate queues, then they can be dispatched to the respective service station **102** in the next stage of operation.

The dispatch process will be described with reference to FIGS. **13** to **17**.

FIG. **13** shows an embodiment of the dispatch process involving throttled and gracefully throttled queues only.

Step **600**: Initiate

At step **600**, the dispatch process is initiated. The method then proceeds to step **602**.

Step **602**: Is Service Station Ready?

At step **602**, it is determined whether the service station is ready to receive a new service request, i.e. if it has finished servicing the previous request the service station **102** has been handling. If not, the method proceeds to step **604**.

If the service station is ready to service a new request, the method proceeds to step **606**.

Step **604**: Wait

At step **604**, the method waits for a predetermined time to enable the service station to complete the current request and be ready to receive a new request. This time period may be any suitable time period.

Step **606**: Throttled Service Request Detected?

The request scheduler **104** examines the service deadline $d_n(S_x, L_y)$ for each throttled queue, starting with the highest priority queue (in this example, the leftmost queue in FIG. **3**), i.e. the queue having the highest priority M. If the deadline for the highest priority throttled queue has not yet been reached, the request scheduler **104** moves to the next throttled queue in line.

This process continues until a throttled queue is reached which has a deadline which has been reached or has passed. At this point, the method proceeds to step **608**.

If, however, the request scheduler **104** reaches the end of the throttled queues and none of the queues has a deadline which has passed, then the method proceeds to step **612**.

Step **608**: Serve Queue for which Deadline has been Reached or Passed

If, in step **606**, it is detected that a queue has a deadline which has been reached or has expired, then the request scheduler **104** will service first request $R_n(S_x, L_y)$ in line in that queue. Since the examining step **606** is carried out in priority order, in practice this means that requests $R_n(S)$ in the throttled queues are serviced starting with the throttled queue having the highest priority for which the deadline $d_n(S_x,L_y)$ for servicing has been reached or has passed.

When a queue is serviced, the request scheduler **104** passes the respective I/O request $R_n(S_x, L_y)$ at the head of that particular queue to the service station **102** for processing. The method then proceeds to step **610**.

Step **610**: Set New Deadline for Serviced Queue

At step **610**, once the throttled queue has been serviced in step **608**, a new deadline is set by which time the next request $R_n(S_x, L_y)$ in the FIFO queue is to be serviced. The new deadline $d_n(S_x, L_y)$ to be set is dependent upon the nature of the system and the required throughput.

The simplest case is for the configuration shown in FIG. **2**. In this arrangement, a service having service ID S_x originating from a single location having location ID L_y sends requests to a single service station **102**. In this particular case, the location ID L_y is not needed. Therefore, it may optionally be omitted from the service request R. Alternatively, if the location ID L_y is included in the service request R, it can be effectively ignored by the service station **102** because there will only be one location ID L_y in each queue.

For the configuration of a single service sending requests from a single location to a single service station, the deadline $d_n(S_x)$ (where y is a constant) is then set based on the I/O size $w_n(S_x)$ of, and service throughput metadata $T(S_x)$ associated with, the next throttled request $R_n(S_x)$ in line in the relevant queue. Therefore, the deadline is set according to equation 2) above.

In other words, the new deadline is the current time plus the time required to process a given amount of data in a request at a required data rate (or bandwidth). Meeting this deadline will mean that the required service throughput is achieved.

In an alternative configuration, a service having service ID S_x and a single location ID L_y may send requests to multiple service stations. In this arrangement, each service station **102** sets deadlines independently of each other service station **102** and there is no communication therebetween.

The new deadline $d_n(S_x)$ for the next request $R_n(S_x)$ set in dependence upon a number of factors. As for the first example, the deadline $d_n(S_x)$ for request $R_n(S_x)$ is dependent upon the I/O size $w_n(S_x)$ and service throughput metadata $T(S_x)$ for the requests having service ID S_x. However, the deadline is set with the aim of achieving aggregated throughput across all of the service stations **102-1-102-***i*, i.e. achieving the requirements set out in the service throughput metadata $T(S_x)$ but in parallel with other service stations **102** processing requests from the same service having service ID S.

The last served distance parameter l is, effectively, the distance of the last served request in the queue for service ID S_x. In other words, for request $R_n(S_x)$ the last served distance l_n would be equal to the distance of the previously served request $R_m(S_x)$ (i.e. the request most recently processed in step **408**), as set out in equation 3) above. The deadline for request $R_n(S_x)$ is, thus, calculated according to equation 4) above.

In other words, the new deadline is the current time plus the time required to process a given amount of data in a request at a required data rate (or bandwidth) in order to achieve an aggregated throughput across all of the service stations **102-***i* to which requests having a particular service ID S have been sent. Meeting this deadline will mean that the required service throughput is achieved.

In the final case, services having a service ID S_x have a number of locations having location IDs L_y, and are serviced by a number of service stations **102**. This is done by utilising a deadline for a particular request which is dependent upon the previous requests from a particular service ID S_x and for a particular location ID L_y, i.e. the "distance" from the earlier requests from that service ID S_x and for that location ID L_y. In this way, an aggregated minimum bandwidth can be guaranteed without administration from a central server attempting to handle distribution across a plurality of service stations to meet a guaranteed bandwidth.

In a sequence of incoming requests for a particular service ID S having location ID L, the distance $x_n(S_x, L_y)$ for the n^{th} request is calculated from equation 8) above.

Knowing the time in which these requests need to be serviced (i.e. the I/O sizes divided by the service throughput T(S1)) enables a deadline system which is able, on aggregate, to meet bandwidth requirements across the distributed system and to meet these requirements fairly across all of the locations of the service having service ID S1. Where l_{in} is the last served distance for request $R_n(S_x, L_y)$ which is equal to the sum of the weights of requests $R_0(S_x, L_y)$ to $R_m(S_x, L_y)$ as set out in equation 9) above. The deadline for request $R_n(S, L)$ is, thus, calculated according to equation 10) above.

Therefore, in this embodiment, the deadlines for each service ID S_x having location ID L_y are effectively independent between location IDs. In other words, within a single queue, the deadline for request $R_n(S1, L1)$ is dependent upon the distance of the most recently-served request $R_m(S1, L1)$ having the same location ID L1. Other requests in the same queue having different location IDs do not affect the deadline for request $R_n(S1, L1)$.

Once the new deadline has been set, the method then proceeds back to step **602**.

Step **612**: Gracefully Throttled Service Request Detected?

Step **612** is reached if, at step **606** it is determined that, in a stepwise scan of each of the throttled queues, none of the throttled queues has a service deadline which has been reached or is expired, then the request scheduler **104** is operable to move to the gracefully throttled queues.

The gracefully throttled queues are, essentially, of lower priority than the throttled queues and are provided to utilise (or "soak up") bandwidth that remains unused once the minimum bandwidth requirements have been met. In practical terms, a throttled connection will be, for example, a paid-for connection where a user pays for a particular amount of storage and a particular guaranteed bandwidth, and a gracefully throttled connection may be one where, at higher cost, extra bandwidth may be used when available.

Consequently, given their low priority, gracefully throttled queues are only serviced if there is currently no throttled queue which is at or has exceeded its respective deadline.

If a gracefully throttled queue is not detected, the method proceeds back to step **604**. However, if a gracefully throttled queue is detected, the gracefully throttled queue to which the service marker is associated is serviced in step **614**.

Step **614**: Service Gracefully Throttled Queue

The gracefully throttled queue to which the service marker is associated is serviced. When a gracefully throttled queue is serviced, the request scheduler **104** passes the respective request $R_n(S_x, L_y)$ at the head of that particular queue to the service station **102** for processing.

Step **616**: Move Service Marker

Once the gracefully throttled request has been serviced in step **614**, the service marker identifying the next gracefully throttled queue to be serviced is moved to the next gracefully throttled queue. Any suitable selection criterion may be used; for example, the next queue in priority order, a round-robin selection, a random queue selection or some other mechanism to select an arbitrary queue for processing.

The method then proceeds back to step **602**.

FIG. **14** illustrates an alternative embodiment whereby, instead of gracefully throttled connections, unthrottled connections are used. The steps of this method are set out below. Steps **700-716** are substantially similar to those of the steps **610-616** and procedures and subject matter in common will not be described again here. Only steps which are different will be described here.

Step **712**: Unthrottled Service Request Detected?

Step **712** is reached if, at step **706** it is determined that, in a stepwise scan of each of the throttled queues, none of the throttled queues has a service deadline which has been reached or is expired, then the request scheduler **104** is operable to move to the unthrottled queues.

The unthrottled queues are, essentially, of lower priority than the throttled queues and are provided to utilise (or "soak up") bandwidth that remains unused once the minimum bandwidth requirements have been met for the throttled connections.

If an unthrottled queue is not detected, the method proceeds back to step **704**. However, if an unthrottled queue is detected, the unthrottled queue to which the service flag is associated is serviced in step **714**.

Step **714**: Service Unthrottled Queue

In step **714**, the unthrottled queue to which the service marker is associated is serviced starting. When the unthrottled queue is serviced, the Request scheduler **104** passes the respective request $R_n(S_x, L_y)$ at the head of that particular queue to the service station **102** for processing.

Unthrottled requests have no throughput requirements. However, unthrottled service requests need to be served fairly and any service of unthrottled requests should not affect the service throughput of throttled requests.

Step **716**: Move Service Marker

The service marker is then moved to the next unthrottled queue in line. This applies until the Request scheduler **104** reaches the end of the unthrottled queues (i.e. the rightmost queue shown in FIG. **11**), the service marker is returned to the first unthrottled queue in line.

The method then proceeds back to step **702**.

Two additional variations of the above method are illustrated in FIGS. **15** and **16**. These illustrate the general approach shown in FIG. **11** whereby throttled connections are provided, together with gracefully throttled connections and unthrottled connections.

The gracefully throttled and unthrottled connections can be provided in any suitable priority order. It is envisaged that gracefully throttled services could take precedence over unthrottled service (as shown in FIG. **15**) and vice versa (as shown in FIG. **16**).

FIG. **15** illustrates a method whereby, if spare capability exists after throttled connections have been served, then gracefully throttled services are served followed by unthrottled services. The steps of this method are set out below. Steps **800-816** are substantially similar to those of the steps **610-616** and procedures and subject matter in common will not be described again here. Only steps which are different will be described here.

Step **812**: Gracefully Throttled Service Request Detected?

Step **812** is reached if, at step **806** it is determined that, in a stepwise scan of each of the throttled queues, none of the throttled queues has a service deadline which has been reached or is expired, then the request scheduler **104** is operable to move to the gracefully throttled queues.

Given their low priority, gracefully throttled queues are only serviced if there is currently no throttled queue which is at or has exceeded its respective deadline. If a gracefully throttled queue is detected, the first gracefully throttled queue in priority order is serviced in step **814**.

However, if a gracefully throttled queue is not detected, in this embodiment the method proceeds to step **818**.

Step **818**: Unthrottled Service Request Detected?

Step **818** is reached if, at step **812** it is determined that, in a stepwise scan of each of the gracefully throttled queues, none of these queues requires servicing. Then, in step **818**, the request scheduler **104** is operable to move to the unthrottled queues.

If an unthrottled queue is detected, the first unthrottled queue in priority order is serviced in step **820**. However, if an unthrottled queue is not detected, the method proceeds back to step **802**.

Step **820**: Service Unthrottled Queue

In step **820**, the unthrottled queue to which the service marker is associated is serviced. When the unthrottled queue is serviced, the Request scheduler **104** passes the respective request $R_n(S_x, L_y)$ at the head of that particular queue to the service station **102** for processing.

Unthrottled requests have no throughput requirements. However, unthrottled service requests need to be served fairly and any service of unthrottled requests should not affect the service throughput of throttled requests.

Step **822**: Move Service Marker

The service marker is then moved to the next unthrottled queue in line. This applies until the request scheduler **104** reaches the end of the unthrottled queues (i.e. the rightmost queue shown in FIG. **11**), the service marker is returned to the first unthrottled queue in line.

The method then proceeds back to step **802**.

Concomitantly, FIG. **16** illustrates a method whereby if spare capability exists after throttled connections have been served, then unthrottled services are served followed by gracefully throttled services. The steps of this method are set out below. Steps **900-916** are substantially similar to those of the steps **700-716** and procedures and subject matter in common will not be described again here. Only steps which are different will be described here.

Step **912**: Unthrottled Service Request Detected?

Step **912** is reached if, at step **906** it is determined that, in a stepwise scan of each of the throttled queues, none of the throttled queues has a service deadline which has been reached or is expired, then the request scheduler **104** is operable to move to the unthrottled queues.

If an unthrottled queue is detected, the method proceeds to step **914** to service the queue. However, if no unthrottled request is detected, the method proceeds to step **918**.

Step **918**: Gracefully Throttled Service Request Detected?

Step **918** is reached if, at step **912** it is determined that, in a stepwise scan of each of the throttled queues, none of the throttled queues has a service deadline which has been reached or is expired, then the request scheduler **104** is operable to move to the gracefully throttled queues.

Given their low priority, gracefully throttled queues are only serviced if there is currently no throttled queue which is at or has exceeded its respective deadline and no unthrottled service request waiting.

If a gracefully throttled queue is detected at step **918**, the first gracefully throttled queue in priority order is serviced in step **920**. However, if a gracefully throttled queue is not detected, in this embodiment the method proceeds back to step **902**.

Step **920**: Service Gracefully Throttled Queue

The gracefully throttled queues are serviced starting with the queue having the highest priority M. When a gracefully throttled queue is serviced, the request scheduler **104** passes the respective request $R_n(S_x, L_y)$ at the head of that particular queue to the service station **102** for processing.

Step **922**: Move Service Marker

Once the gracefully throttled request has been serviced in step **920**, the service marker identifying the gracefully throttled queue to be serviced is moved to the next gracefully throttled queue in line.

The method then proceeds back to step **902**.

A final variation is shown in the embodiment of FIGS. **17** and **18**. In this embodiment, instead of setting a priority order for the servicing of throttled and/or unthrottled queues, a single fair scheduling system is used to address both throttled and unthrottled queues equally. FIG. **17** shows a schematic diagram whereby throttled connections are provided, together with gracefully throttled connections and unthrottled connections. FIG. **18** shows a method according to an alternative embodiment of the disclosure.

The steps of this method are set out below. Some of steps **1000-1016** are substantially similar to those of the steps **610-616** as set out in FIG. **13** and procedures and subject matter in common will not be described again here. Only steps which are different will be described here.

In the method described below, gracefully throttled and unthrottled queues are treated equally without a preferred priority order. Therefore, a common service marker is shared between these queues and is moved as appropriate after a queue has been serviced.

Step **1012**: Gracefully Throttled or Unthrottled Service Request Detected?

Step **1012** is reached if, at step **1006** it is determined that, in a stepwise scan of each of the throttled queues, none of the throttled queues has a service deadline which has been reached or is expired, then the request scheduler **104** is operable to move to the gracefully throttled queues and the unthrottled queues.

As previously described, the gracefully throttled queues and unthrottled queues are able to utilise bandwidth that remains unused once the minimum bandwidth requirements have been met for the throttled queues. Consequently, given their low priority, gracefully throttled queues and unthrottled are only serviced if there is currently no throttled queue which is at or has exceeded its respective deadline.

If neither a gracefully throttled queue nor an unthrottled is detected, the method proceeds back to step **1004**. However, if a gracefully throttled queue or an unthrottled queue is detected, the gracefully throttled queue or unthrottled queue to which the service marker is associated is serviced in step **1014**.

Step **1014**: Service Gracefully Throttled Queue or Unthrottled Queue

The gracefully throttled queue or unthrottled queue to which the service marker is associated is serviced. When a gracefully throttled queue or an unthrottled queue is serviced, the request scheduler **104** passes the respective request $R_n(S_x, L_y)$ at the head of that particular queue to the service station **102** for processing.

The service marker in this example is common between both gracefully throttled queues and unthrottled queues. In other words, in contrast to previous embodiments that had a separate service marker for each queue type, in this arrangement, a single service marker is shared between both types of queue.

Step **1016**: Move Service Marker

Once the gracefully throttled request or an unthrottled request has been serviced in step **1014**, the service marker identifying the next gracefully throttled or unthrottled queue to be serviced is moved to the next gracefully throttled or unthrottled queue. Any suitable selection criterion may be used; for example, the adjacent queue, a round-robin selection, a random queue selection, an alternate gracefully throttled to unthrottled (or vice versa) selection or some other mechanism to select an arbitrary queue for processing.

The method then proceeds back to step **1002**.

The method outlined above enable service throughput for a throttled service to be guaranteed. In other words, if the requests $R_n(S_x)$ are arriving at a higher rate than the specified bandwidth (or throughput) they need to be slowed down before being dispatched to the service station **102** by the Request scheduler **104**. On the other hand, if the requests $R_n(S_x)$ are arriving slowly it is not necessary to slow them down. They can be dispatched to an available service station immediately.

In the above-described method, it is possible for the status of a service to change at any time. For example, a throttled service could become an unthrottled service or vice versa. This can be achieved by changing the service throughput metadata T associated with the request R as appropriate.

In this situation, a particular queue associated with a particular service ID S can be moved from the throttled queue section to the unthrottled queue section at any point based upon the updating of the service throughput metadata T(S) sent to the Request scheduler **104**. The advantage of this arrangement is that a particular request $R_n(S_x)$ itself does not need to change format in order to move from one service type to another. This reduces the amount of processing and data exchange required.

The above scheduling method has a number of advantages. Throughput (i.e. bandwidth) is guaranteed for the throttled service when the system operates within its capacity. Further, when the system is operating within its capability, no throttled or unthrottled service can disrupt the throughput of other throttled or unthrottled services by send requests at a higher rate. This avoids a major drawback of conventional throttled service arrangements which can only impose a maximum possible bandwidth as opposed to a guaranteed minimum.

Furthermore, the described method enables unthrottled requests to be serviced by the service station **102** as well. The unthrottled requests are serviced only when there is no throttled service requests and therefore do not affect the throughput of the throttled services. Concomitantly, the ability to service unthrottled requests when system capacity permits enables improved utilisation of the service station **102**.

Additionally, throughput is throttled for throttled service queues. Therefore, in this embodiment, for a given service ID S_x the maximum throughput cannot be more than the associated service throughput $T(S_x)$ even when corresponding service requests $R_n(S_x)$ arrive at a higher rate.

The method outlined above enable service throughput for a throttled service to be guaranteed across multiple service stations receiving requests from a distributed service having multiple service locations. The above scheduling method has a number of advantages. Throughput (i.e. bandwidth) is guaranteed for the throttled service across a parallel file system when the system operates within its capacity. This bandwidth is guaranteed without each service station needing to share or communicate deadlines or other request processing information with other service stations. This eliminates a to potentially enormous amount of data communication which would otherwise be required if, for example, a central server managed the data transactions and set the deadlines for each request to be processed.

FIG. **19** is a graph of bandwidth usage on service stage connected to a typical storage network used by a plurality of applications. As shown, when all applications are unthrottled, close to the maximum capacity of the service station is used at times. However, when the method of the present disclosure is used, all applications are maintained at their specified minimum bandwidth, irrespective of the other connections. FIG. **19** illustrates the ability of the present disclosure to manage the bandwidth of multiple services.

Variations of the above embodiments will be apparent to the skilled person. The precise configuration of hardware and software components may differ and still fall within the scope of the present disclosure. For example, the location ID aspect of the third embodiment above could be equally applied to the single service station configuration of the first embodiment.

Whilst the distance and last served distance aspects above have been set out in terms of summing from 0 to n for the n^{th} request, it is also possible to sum from 0 to n−1 and then add in the weight of the n^{th} request, if desired.

Alternatively, if the requests are all of the same weight (i.e. I/O size), then the request number n or request count can be used to determine the distance (and, thus, the deadline) on the service station side.

Additionally, the distance parameter may be sent in a different format. For example, the distance, x_n, could be sent as a function which is then processed by the service station. Examples of such functions may be αx_n where α is a constant or other function. Alternatively, function such as $(x_n)^2$ or $(x_n)^3$ may be used.

A more general approach may be to provide a general function, such as, for example, F, where F is an invertible function that has inverse G. Then, the request may comprise $F(x_n(R_n))$ from the client or service origin side to obtain $x_n(R_n)$ by inversion at the server/service station side because $x_n(R_n)=G(F(x_n(R_n)))$. Indeed, the examples given above Xn(Rn) itself corresponds to function F(x)=x which has inverse G(x)=x. A similar function could be used for the example given above in the $x_n(R_n)=n$ case where all requests have equal weight.

In general, the distance parameter provides information identifying how much I/O has been sent to other service stations before sending a request. As set out above, the distance parameter may be supplied with the request or may be calculated from the supplied request number n.

However, variations are possible. For example, the service throughput T may, optionally, be supplied with the request rather than supplied by other means. Therefore, the request may optionally comprise the deadline data itself, i.e. the distance of the particular request divided by the throughput required for the service ID associated with the request. Alternatively, the deadline could be calculated on the service station side from the distance and the throughput data provided separately in the same request.

In addition, with regard to gracefully throttled services, the gracefully throttled identifier metadata may be supplied by the service coordinators to the service stations as described in the manner of the service throughput metadata T. The gracefully throttled identifier metadata may also be supplied as part of the service throughput metadata T. As a further variation, the gracefully throttled identifier may be included in the service request if desired.

In general, a number of permutations for determining deadlines may be used. For example, as set out above, a request may comprise only a request number, and the distance is calculated based on constant weights at the service station and the deadline determined from the throughput data received separately. Alternatively, the request may comprise a distance metric, with the throughput data being received separately by the service station and the distance/deadline determined at the service station

Further, the request may comprise distance data determined at the service origin or client side and throughput data received separately by the service station. Finally, the request may comprise both distance data and throughput data either separately or in the form of deadline data sent to the service station. The final arrangement does not require the service station to perform any determination or calculations to determine the deadline. This may be beneficial in some configurations.

In the examples illustrated above, each service station may, for example, comprise an object storage server (OSS) that stores file data on one or more object storage targets (OSTs). Typically, an OSS serves between two and eight OSTs, with each OST managing a single local disk filesystem (not shown). The capacity of the file system is the sum of the capacities provided by the OSTs **8**.

The service stations may, optionally, be connected to a metadata server (MDS). The MDS 3 has a single metadata target (MDT) per filesystem that stores metadata, such as filenames, directories, access permissions, and file layout. The MDT data is stored in a single local disk filesystem MDS. The MDS is conFIG.d to function as a portal for a client computers or services to the parallel file system comprising the storage located on the plurality of service stations. This may take the form of, for example, a webpage or a portal whereby a user ca request access to a data storage resource.

The present disclosure is particularly applicable to parallel distributed file systems. These systems are often used for large scale cluster computing and may comprise file systems such as Lustre™ or Collibri™. However, it will be appreciated that this need not be the case and that other distributed network systems may fall within the scope of the present disclosure.

Embodiments of the present disclosure have been described with particular reference to the examples illustrated. While specific examples are shown in the drawings and are herein described in detail, it should be understood, however, that the drawings and detailed description are not intended to limit the disclosure to the particular form disclosed. It will be appreciated that variations and modifications may be made to the examples described within the scope of the present disclosure.

What is claimed is:

1. A system comprising:

at least one server coupled to a pool of storage devices to transfer data from the storage devices to client devices responsive to requests; and

a request scheduler adapted to receive into a memory a plurality of requests each comprising a service identifier (ID) and a payload size, to set a deadline for each request responsive to the service ID and the payload size, to forward the requests to the server for processing in an order based on service ID and, responsive to the deadline being reached for a selected request, to advance the selected request for immediate processing by the server.

2. The system of claim **1**, the service ID for the selected request specifying a required minimum data transfer rate, the request scheduler establishing the deadline for the selected request to meet the required minimum data transfer rate.

3. The system of claim **1**, the request scheduler further adapted to sort the received requests into a throttled queue and an unthrottled queue responsive to the service IDs, the requests in the throttled queue having a required minimum data transfer rate, the requests in the unthrottled queue having no required minimum data transfer rate, the requests in the throttled queue receiving priority over the requests in the unthrottled queue.

4. The system of claim **1**, the request scheduler further setting the deadlines for the requests based on multiple received requests having the same service ID.

5. The system of claim **1**, the request scheduler further setting the deadlines for the requests based on an elapsed time interval between successively received requests having the same service ID.

6. The system of claim **1**, further comprising a plurality of first-in-first-out (FIFO) queues into which the request scheduler sorts the received requests based on service IDs.

7. The system of claim **6**, each of the FIFO queues storing requests sharing a common service ID and received from a common client device.

8. The system of claim **7**, the FIFO queues further sorted by priority levels associated with the service IDs therein so that a request is forwarded for processing by the at least one server from each FIFO queue in turn.

9. The system of claim **1**, the request scheduler further determining the deadlines based on a distance parameter relating to a sum of the payload sizes of a plurality of requests sharing a common service ID.

10. The system of claim **1**, the request scheduler generated an updated deadline for a particular request responsive to a previously processed request sharing a common service ID with the particular request.

11. The system of claim **1**, the request scheduler setting a deadline for an nth request having a particular service ID in a sequence of n requests sharing a common service ID based on a sum of the payload sizes for the n requests.

12. A storage system comprising:

a storage server comprising a programmable processor and associated memory;

a plurality of storage devices coupled to the storage server adapted to store data objects from client devices coupled to the storage server;

a memory adapted to accumulated pending client requests for data objects from the client devices; and

a request scheduler adapted to manage the pending client requests in the memory for processing by the storage server, the request scheduler comprising a programmable processor and associated memory adapted to establish a deadline for each pending client request responsive to a service ID value and a payload size value associated with the client request, to sort the pending client requests into a plurality of first-in-first out (FIFO) queues based on the service ID values processing in an order based on service ID and, to forward the queued client requests from the FIFO queues to the storage server, to monitor the deadlines and to advance a selected queued client request for immediate execution by the storage server responsive to the associated deadline being met.

13. The storage system of claim **12**, the service ID values for at least a portion of the pending client requests specifying required minimum data transfer rates, the request scheduler establishing the deadlines for the at least a portion of the pending client requests to meet the required minimum data transfer rates.

14. The storage system of claim **13**, the request scheduler sorting the portion of the pending client requests into throttled FIFO queues, the request scheduler sorting a remaining portion of the pending client requests without specified required minimum data transfer rates into at least one unthrottled FIFO queue.

15. The storage system of claim **12**, the request scheduler further setting the deadlines for the client requests based on multiple received client requests having the same service ID.

16. A computer-implemented method comprising:

storing data objects from client devices in a pool of data storage devices using at least one storage server;

accumulating, in a memory, client requests to transfer the data objects to a number of client devices, each client request having a service ID value and a payload size value;

determining a deadline for each client request responsive to the service ID value;

forwarding the client requests to the at least one storage server for processing in an order based on the service ID values; and

monitoring the deadlines and, responsive to the deadline being reached for a selected client request, advancing the selected client request for immediate processing by the at least one storage server.

17. The method of claim **16**, the service ID for the selected client request specifying a required minimum data transfer rate, the request scheduler establishing the deadline for the selected client to meet the required minimum data transfer rate.

18. The method of claim **16**, further comprising sorting the client requests into at least one throttled queue and at least one unthrottled queue responsive to the service ID values, the client requests in the throttled queues having a required minimum data transfer rate, the client requests in the unthrottled queues having no required minimum data transfer rate, the client requests in the throttled queues receiving priority over the client requests in the unthrottled queue.

19. The method of claim **16**, the deadlines for the client requests further based on an elapsed time interval between successively received client requests having the same service ID value.

20. The method of claim **16**, further comprising generating an updated deadline for a particular client request responsive to a previously processed client request sharing a common service ID value with the particular client request.

* * * * *